Aalto University
School of Science
Degree Programme in Computer, Communication, and Information Science

Binda Pandey

Adaptive Learning For Mobile Network Management

Master's Thesis
Espoo, November 2, 2016

Supervisor: Docent Tomi Janhunen, Aalto University
Advisor: Jussi Rintanen D.Sc. (Tech.)

Aalto University
School of Science
Degree Programme in Computer, Communication, and Information Science

ABSTRACT OF MASTER'S THESIS

Author:	Binda Pandey		
Title:	Adaptive Learning For Mobile Network Management		
Date:	November 2, 2016	**Pages:**	vii + 54
Major:	Computer Science	**Code:**	SCI3042
Supervisor:	Docent Tomi Janhunen		
Advisor:	Jussi Rintanen D.Sc. (Tech.)		

There is an increasing demand for better mobile and network services in current 4G and upcoming 5G networks. As a result, mobile network operators are finding difficulties to meet the challenging requirements and are compelled to explore better ways for enhancing the network capacity, improving the coverage, as well as lowering their expenditures. 5G networks can have a very high number of base stations, and it costs time and money to configure all of them optimally by human operators. Therefore, the current network operations management practice will not be able to handle the network in the future. Hence, there will be a need for automation in order to make the network adaptive to the changing environment.

In this thesis, we present a method based on *Reinforcement Learning* for automating parts of the management of mobile networks. We define a learning algorithm, based on Q-learning, that controls the parameters of base stations according to the changing environment and maximizes the quality of service for mobile devices. The learning algorithm chooses the best policy for providing optimal coverage and capacity in the network. We discuss different methods for taking actions, scoring them, and abstracting the networks' state.

The learning algorithm is tested against a simulated LTE network and the results are compared against a base-line model, which is a fixed-TXP model with no control. We do the experiments based on above-mentioned strategies and compare them in order to evaluate their impact on learning.

Keywords:	Mobile Networks, Reinforcement Learning, Q-learning, Network Management, Artificial Intelligence, Intelligent Control
Language:	English

Acknowledgements

I would like to express my deep gratitude to my supervisor Docent Tomi Janhunen and my advisor Dr. Jussi Rintanen for giving me this opportunity to work in a challenging yet an interesting project that I present here in my thesis. I would like to thank Tomi for his moral support, guidance and providing the proper environment that helps me in writing this thesis in time.

I also would like to thank Jussi for his immense help and support. He has helped me greatly with the implementation part of this project by providing ideas and correcting my mistakes. Moreover, I would appreciate him for providing his valuable time in examining my thesis drafts regularly and carefully that has helped me a lot to carry out my thesis work.

I further wish to extend my gratitude to Dr. Vilho Räisänen and Juha Niiranen from Nokia Bell Labs for their collaboration and for defining the problem in the mobile networks which has been the motivation for this work. Vilho has provided the concept of the simulated network and also designed a mini-simulator that has helped a lot for the early design of our algorithm. Moreover, he has given his valuable time by running all the experiments in their simulated network and giving some feedback on them as well. I would also like to thank Juha for running some simulations and providing ideas on the working nature of the simulator.

Apart from this, I am grateful to the technical staff at the Department of Computer Science for their assistance and support. I also would like to thank Aalto University for providing computational resources.

Finally, I wish to thank all of my friends who have supported me in any way during this tenure.

Espoo, November 2, 2016

Binda Pandey

Abbreviations and Acronyms

1G	First Generation
2G	Second Generation
3G	Third Generation
3GPP	3rd Generation Partnership Project
4G	Fourth Generation
5G	Fifth Generation
ANR	Automatic Neighbor Relations
AI	Artificial Intelligence
BTS	Base Transceiver Station
CAPEX	Capital Expenditure
CCO	Coverage and Capacity Optimization
CM	Configuration Management
CQI	Channel Quality Indicator
DSM	Dynamic Spectrum Management
DSP	Digital Signal Processing
EDGE	Enhanced Data Rates for GSM Evolution
FM	Fault Management
GSM	Global System for Mobile Communications
ICIC	Inter-Cell Interference Coordination
IOT	Internet of Things
KPI	Key Performance Indicator
LTE	Long Term Evolution
M2M	Machine to Machine Communications
MLB	Mobility Load Balancing
MMS	Multimedia Message
MRO	Mobility Robustness Optimization
NE	Network Element
NGMN	Next Generation Mobile Network
NMS	Network Management System
OAM	Operation and Maintenance

OMC	Operation and Maintenance Centre
OPEX	Operational Expenditure
PCI	Physical Cell ID
PM	Performance Management
POMDP	Partially Observable Markov Decision Process
QoS	Quality of Service
RET	Remote Electrical Tilt
RL	Reinforcement Learning
RLF	Radio Link Failure
SNR	Signal to Noise Ratio
SON	Self Organizing Network
TD	Temporal Difference
TXP	Transmission Power
UE	User Equipment
UMTS	Universal Mobile Telecommunications System

Contents

Chapter 1

Introduction

In the past few years, there has been a rapid increase in the number of mobile subscribers as well as in the use of new data services such as social networking, video streaming, and so on in mobile telecommunications networks. Moreover, recent technologies, such as smartphones and tablets have powerful processors and support high-end multimedia applications. In addition, the concept of Machine to Machine (M2M) communications is emerging rapidly due to automation and Internet of Things (IOT). These all have caused the mobile data traffic to grow exponentially [16] and hence there is a high demand for enhancing the network capacity and improving Quality of Service (QoS).

This leads to the advent of Long Term Evolution (LTE) in the hope of achieving the strong requirements posed by the current situation in cellular networks as well as getting the cost per bit down [16, 40]. LTE refers to the latest standard for high-speed wireless data communication networks which is capable of increasing the network capacity and speed using new digital signal processing techniques (DSP). However, handling the larger data volume in LTE in different cell contexts is a real challenge for network management. The number of base stations will increase and the effort of maintaining them increases as well. Moreover, most of the network management functions such as network planning, configuration, management, healing, and optimization require substantial work effort and high cost, both on capital investment and operational expenditures (CAPEX and OPEX) [16, 40], and handling such functions manually is all the more difficult. Apart from that, manual effort is time-consuming and prone to errors. Thus, manual configuration of each network element becomes gradually impossible with the growing number of network elements. Hence, there is a need for automation of these tasks so that the network operation expenses get minimized and Quality of Service (QoS) gets improved in terms of connection reliability, data bandwidth, and

latency.

Recently, an emerging concept called Self Organizing Network (SON) is being introduced to address these challenges in current 4G and upcoming 5G networks. SON aims at increased automation of network operations in order to utilize network resources in an optimized way. Currently, SON consists of numerous functions for automatic configuration, optimization, and healing of mobile networks [33]. These functions are implementations of use cases and implementing numerous use cases is a cumbersome job. A more pressing problem, however, is tailoring the the behavior of existing SON function types to different cell contexts and coordinating their behavior. Furthermore, global optimization of network is futile, as the situation changes continuously and it is hence better to perform localized optimizations to address specific issues. Therefore, the future goal is to introduce an automated network management function which can learn from data and adapt.

In this thesis, we are investigating adaptive methods for cellular network management at the level of base stations. In particular, the work attempts to find methods for learning useful management policies for SON functions such as adjusting the main transmission parameters of base station antennas, primarily transmission power and antenna tilt. We adopt the framework of reinforcement learning [38] and use the independent learning approach where the network parameters of base stations are adjusted independently without globally optimizing the parameters of all base stations jointly. The approach ignores the network parameters of other base stations. The reason behind choosing the independent approach instead of joint learning is due to the good scalability of independent learning. There is a large number of base stations and optimizing each of their parameters will result in very large state spaces if the situation of other base stations is also considered when adjusting the parameters of one. On the other hand, the state space is manageable if the learning is independent.

The state space of mobile networks has an astronomical size, but can be significantly reduced by abstraction. Hence, one of the goals of this thesis is to design abstraction methods which accurately abstract the network state. We provide various ways of abstracting the given network state and compare them. Our further aim is to define methods for scoring actions in terms their impact on QoS and examine their effects on learning. In addition, we evaluate different exploration/exploitation policies in choosing the best settings in the network and perform experiments based on them as well.

1.1 Structure of the Thesis

The structure of the thesis is as follows. In Chapter 2 we give an overview of mobile networks and network parameters. Here, we focus on current practices for network management. Further, we discuss the difficulties of the current approaches to the management of the current 4G and upcoming 5G networks and conclude the chapter by showing that there is a need for an automated approach for the efficient management of the networks. In Chapter 3 we present the theory behind reinforcement learning in detail. We explain essential elements of reinforcement learning and its types and focus on Q-learning. In Chapter 4 we present our learning algorithm designed for network management including abstraction methods, scoring functions, and exploration and exploitation policies. In Chapter 5 we study the performance of the methods of Chapter 4. In Chapter 6 we discuss issues related to our learning method and also provide possible extensions for future work. We finally give a brief summary of the whole thesis work and make some concluding remarks.

Chapter 2

Background

A mobile network [12] is a wireless communication network which provides radio coverage to large geographical areas. A mobile network user can connect his or her mobile terminal, such as a mobile phone or a tablet, to the network and move within the service area without losing the connection to the network. The continuous coverage is provided by placing base transceiver stations (BTS) at various locations throughout the geographical area of the network. Each base station contains one or more antennas which provide coverage to a limited area, called a *cell*. Figure 2.1 shows a typical layout of base stations covering a part of the mobile network.

There are several types of cells, based on their size and capacity. They are *Macrocells, Microcells, Picocells*, and *Femtocells*. Macrocells provide a wide range of coverage and are found in rural areas or highways. Microcells are of a size of few hundred metres and are suitable for densely populated urban areas. Picocells are used for smaller areas such as offices or shopping centers. Finally, femtocells cover the smallest area and are installed in homes.

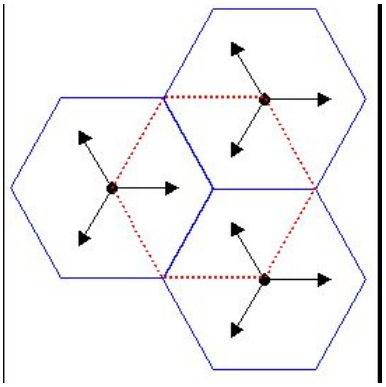

Figure 2.1: Typical layout of a base station

2.1 Network Management

A mobile network is run by a network operator and the network operations are operated through a centralized Operation, Administration, and Maintenance (OAM) architecture. The key element of OAM is Operation and Maintenance Centre (OMC) which is responsible for monitoring the network, configuring the network parameters, and optimizing the network behavior. Network Management System (NMS) is the main part of OMC that operates the network. The operator can monitor and access the network elements (NEs) by using a software system called NMS. The network management includes a larger set of functions which are included in following functional areas [17, 25].

- *Fault management (FM)* provides information on the status of the network. Any malfunctions in the network component are detected and transferred to NMS so that technicians can repair them.
- *Configuration management (CM)* functions deal with network configuration parameters such as frequency plans and handover algorithms which are directed from NMS to all NEs.
- *Performance management (PM)* is responsible for monitoring network performance. The data from all network elements is collected by the NMS for further processing and analysis.
- *Security management* is responsible for handling security measures, such as system logs and access control.

2.2 Network Data

Network elements produce PM and FM data (measurement and alarms) whereas CM data are typically created by NMS (including SON functions). NMS collects these data for further analysis. These data can be divided into three categories, *system configuration data, system parameter data*, and *dynamic data* [17]. System configuration data tells about the configuration and organization of the network such as BTS locations and transmission network topology. These types of data are highly persistent and are not usually modified once the network is deployed. System parameter data defines the way the system functions are implemented and operated within the network. Examples of such data include antenna transmitter output power and

BTS threshold for the maximum number of accepted simultaneous connections. These parameters can be either modified by the network operator or by autonomous processes according to the changes in the traffic conditions. Finally, dynamic data consists of the measurement values and statistical time series, alarms, and other events. These data describe the dynamic use of the network and its functions [17].

The system parameter data is handled by configuration management functions. Parameter changes are defined in the NMS and then deployed to NEs. The events occurring in various NEs are counted and recorded during the operation of the network, thereby generating counters [25]. Time frames like 15 minutes, one hour, or one day, are used in the counting for management purposes. These counter values are analyzed in order to gain information on the network performance. However, the number of counters is huge and it becomes a complex task and impractical to use all of them. Therefore, they are usually aggregated to higher level variables called Key Performance Indicators (KPI) [39]. These KPI will provide an overview of the cell states in the mobile network.

In this work, we concentrate on Radio Link Failure (RLF) and Channel Quality Indicator (CQI). RLF event occurs when a terminal loses its connection to the network. A terminal frequently sends a CQI report to the BTS [12]. CQI reports indicate maximum data rates that the terminals can reach.

2.3 History of Mobile Telecommunication System

2.3.1 From 1G to 3G

First generation systems [12] were based on analog communication techniques introduced in the early 1980s. The networks had very large cells and the radio spectrum was not used efficiently. Thus, the capacity of the network was comparatively small. After a decade, these systems were replaced by 2G digital telecommunications which were more efficient in the use of the radio spectrum. 2G technologies were originally designed for voice and were later enhanced to support data services such as text messages, picture messages, and multimedia messages (MMS). The most popular 2G system was the Global System for Mobile Communications (GSM). With the increase in the data rates over the Internet, 3G systems were introduced using techniques such as *Enhanced Data Rates for GSM Evolution* (EDGE) to facilitate growth, increase bandwidth, and support more diverse applications.

The most dominant 3G system is the Universal Mobile Telecommunication System (UMTS).

2.3.2 LTE and Self Organizing Network

Mobile data traffic started to increase dramatically due to the introduction of new technologies such as Apple's iPhone and Google's Android operating system for mobile phones. These smartphones support many mobile applications that require Internet with higher data rates [12]. The problem with data growth in 3G is too high cost per bit. Hence, there was an urgent need to upgrade the network. Thus, the concept of LTE was introduced. LTE is a 4G communication standard which evolved from the 3G technology and is an enhancement of UMTS (Universal Mobile Telecommunication Systems). LTE is faster than UMTS (one of the 3G systems) and thus supports features such as a high data rate, low transmission latency, lower cost per bit, and packet optimized radio access networks [3, 16]. LTE also provides much greater spectral efficiency than the most advanced 3G networks.

However, the configuration of cells for large data volumes becomes problematic in LTE. The configuration, planning, and optimization of network elements are done by human operators and this becomes infeasible with the increase in the number of base stations. Thus, the concept of Self Organizing Networks (SON) becomes an integrated part of LTE which aims at reducing expenses by automating the network operations. SON has a self-organizing capability that enables the network to configure itself, thereby reducing the complexity, cost, and time. SON consists of a set of functions which are defined by the 3^{rd} Generation Partnership Project (3GPP) and the Next Generation Mobile Network (NGMN) alliance [24] and, can be categorized into three functional areas.

- *Self-Configuration* is the process of providing "plug and play" functionality in network elements which minimizes the human intervention in the overall installation process when new components are added into a network. It involves several distinct functions such as automatic software management, physical cell ID configuration (PCI) and automatic neighbor relations (ANR) [33]. Self configuration should take care of the tasks such as bringing new NEs or NE parts, automatic connectivity setup between the NE and the OAM system, as well as automatic configuration of BTS radio parameters.

- *Self-optimization* functions aim at maintaining the quality and performance of the network with minimum manual intervention. These

functions analyze the performance of network elements and automatically trigger optimal actions. Automatic neighbor relations (ANR), inter-cell interference coordination (ICIC), mobility load balancing optimization (MLB), and mobility robustness optimization (MRO) are some of the important self-optimization SON functions [33]. ANR is used both in self-configuration and optimization. Neighbor relations need to be set up during the configuration of each NE. In addition, they need to be maintained during run time because of the change in coverage areas and handover behavior.

- *Self-healing* functions aim at detecting malfunctions and resolve or mitigate them, thereby reducing maintenance costs [33].

Network coverage and capacity is one of the important objectives in mobile networks and is also the main focus of this thesis. This objective is achieved by one of the important SON functions called Coverage and Capacity Optimization (CCO) [24]. The coverage and capacity of a cell may vary due to changes in the environment. The changes can be due to season, such as trees in full leaf during summer, heavy snowfall during winter, or man-made changes in the environment. The requirements in coverage and capacity may also change due to variations in traffic distribution, for example in city centre during rush hour. CCO adjusts the transmission power and antenna tilt to avoid any gaps in coverage. However, unnecessary overlapping between cells might lead to interference. Therefore, CCO needs to adjust the transmission parameters so that the coverage is good as well as the interference is minimized [16].

SON functions reduce human effort in managing a network. However, interference between these functions may lead to coordination problems [24, 34]. For example, a conflict occurs when two instances of a CCO function change the parameters of two neighboring sectors simultaneously [24].

Individual SON functions perform well in separation. However, in joint operation, the impact on the network performance is not always the best. To mitigate such coordination problems, the dependencies among SON functions have to be reduced. But, it becomes infeasible to fix all possible conflicts manually as the number of SON functions increases.

2.4 Need for Intelligent Automation

As discussed above, the SON functions are capable of optimizing and maintaining the mobile network. In the existing system, human operators define all the objectives in the form of parameters and policies for configuring the

behavior of SON functions. These parameters and policies depend on the network environment and the technical properties of the network. SON uses the rule set defined by human operators and has difficulties in adapting the changing network on its own. New rules need to be defined to cope up with every specific network state which is not feasible as the network grows large. In addition, optimizing the configurations for many cells having different characteristics (different numbers of users, different data usage patterns) is another problem and demands more manpower. The future networks adapting 5G technology will be highly dynamic and will have frequent changes in the operational context. Therefore, SON functions need to be highly intelligent and possibly automated so that the cells can adjust their settings on their own according to the changing environment. Thus, AI technologies and machine learning need to be adapted to make the SON functions more intelligent and less dependent on human operators [16].

2.5 Related Work on Automation of Network Management

Much research has been done regarding the automation of mobile networks. In [40], an automated method was proposed to deal with self-healing functions. The method had two main steps: *detection* and *diagnosis*. Detection of an anomaly was done by monitoring the performance indicators (KPI) and comparing them to their usual behavior. In diagnosis, the reports of previous fault cases were looked up and best matching root causes were identified.

Räisänen et al. [10] have presented a demonstrator system that is capable of detecting network failures and recovering from them automatically. They defined two steps to solve this task : *anomaly analysis* and *recovery analysis*. In anomaly analysis, abnormal cells were detected by using user measurement reports. Data mining approach (k-nearest neighbor) was used to classify a cell as normal or abnormal. In the second stage, solutions are proposed based on an anomaly report. They used a case based reasoning algorithm to select the best recovery actions.

Chapter 3

Reinforcement Learning

3.1 Introduction

Reinforcement learning [38] is a form of machine learning where an agent learns to interact with an unknown environment in order to maximize its numerical reward. The reward is the feedback obtained from the environment whenever an action is taken. The agent learns to act and optimize its behavior from the feedback obtained from its actions [21]. That means, for the first time, if an agent takes an action and receives a bad response for it from the environment, then from the next time onwards it learns not to choose the same action in that state. For example, a child who is burned by fire will learn not to go near fire.

Figure 3.1 is a general architecture of RL that depicts agent-environment interaction. The agent takes an action that changes the state of the environment. Subsequently, the environment gives feedback in the form of a reward. The reward can either be positive or negative. It is a continuous process where an agent receives a new state (s_t) and reward (r_t) for each time step t and gradually learns to take the best action (a_t). The utility of the agent is defined by the reward function, and the agent must learn to act to maximize the expected reward.

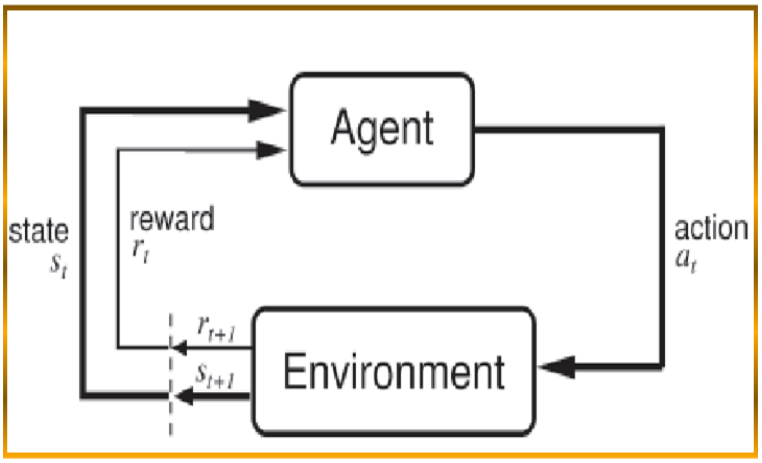

Figure 3.1: Agent-Environment Interface in Reinforcement learning

Reinforcement learning is different from supervised learning. Supervised learning is learning from examples where the examples and the associated feedback are provided by experts. However, in reinforcement learning the agent is not trained with training examples and hence is not explicitly told about the difference between good and bad actions. By good actions, we mean the actions that lead to high rewards in the long run and by bad actions, we mean the actions that lead to negative rewards or costs. Hence, the objective of reinforcement learning is to make the agent intelligent so that it can discover the best actions from its own experience. In order to achieve this, the agent should learn to recognize good and bad actions by trial and error. The next important thing about choosing actions is that, in many cases some actions give a high immediate reward but in the long run, fail to maximize the long term reward. But, the objective of reinforcement learning is always about choosing the actions that maximize the long term reward. Some examples of reinforcement learning are the following.

- When learning to crawl, any forward motion can be considered a positive reward [36].
- In a chess game, when you checkmate the opponent, then it is a good action, and being checkmated by the opponent indicates something bad has happened and your chance of losing the game is high.
- The N-arm bandit problem [23] is a reinforcement learning problem. Here, the player has n arms to pull and receives initially an unknown and possibly stochastic payout. Hence, the player has to learn to choose the arm that has the highest payouts.

The state space and actions in reinforcement learning are known whereas transition probabilities between states and the reward function are unknown.

3.2 Elements of Reinforcement Learning

There are four main components in reinforcement learning. They are *a policy, a reward function, a value function* and *a model* of the environment [38].

Policy Policy is stochastic mapping from states to actions. In RL, the learning behavior is determined by the agent's chosen policy. Agent observes the state and chooses an action accordingly. Policy can be a simple function or a lookup table in some cases or it may involve a extensive computation in others.

Reward Function The reward function is a mapping of the state of the environment to a numerical value. The function should distinguish between good and bad events for the agent. It indicates the immediate reward for actions.

Value function Reward function gives immediate payoff for a given state and hence is not sufficient for decision making in the long term. Hence, value function which gives long-term payoff of a state comes into play. A state with low reward can still have a high value because it could be followed by other states that yield high rewards.

Model This is an optional element in reinforcement learning. One can first learn the transition probabilities among all the states and the reward function for each of the states, and this information can be used to build an explicit model of the system. Such models can be used for planning where the future situations are taken into account while deciding any actions in current situation [38]. Methods like dynamic programming [4, 5] are used in model-based systems in order to learn optimal value functions and to find optimal policies from them [31]. There are many other model-free (direct) learning techniques such as Q-learning [47] and TD(λ)-methods [38] which learn the state value function (or the Q-value function) directly (without the need of any explicit model) and obtain an optimal policy.

Both model-based and model-free RL methods have their own pros and cons depending on the type of problem. Atkeson et al. [2] compared model-based and model-free learning on a simple task called *pendulum swing up*

and observed that model-based learning is more efficient. Similarly, Wiering [49] has used a model-based RL algorithm on traffic light control. He used a model-based technique for estimating cumulative waiting time of cars. On the other hand, research with model-free RL techniques has also been done. Lauer and Riedmiller [26] have proposed a model-free distributed Q-learning algorithm in cooperative multi-agent systems to find optimal policies. Similarly, Littman [29] has provided a Q-learning method for finding optimal policies in mutliplayer games such as soccer. Other research on model-free Q-learning technique includes [22, 41].

We will be using the model-free technique for mobile network management problem and will be focusing on Q-learning only.

3.3 Q-Learning

Q-learning [47] is a widely used model-free reinforcement learning technique where the Q-value of each action a in a given state s in maintained in a table. In other words, it learns the action-value function which gives the expected utility of taking an action in a given state and thereafter following the best policy. The convergence of Q-learning to the optimum action-values has been proved under certain assumptions [46]. The values for each state and action are updated every time by an update function. Hence, the learning is on-line and there is no need for an explicit transition model. The general procedure for Q-learning techniques is given as Algorithm 1.

Algorithm 1: Q-learning

```
1 Initialize Q(s,a) with some arbitrary value (for example, 0).
2 Let the current state of an agent be s.
3 while True do
4 |   a ⟵ Action chosen by agent at time t derived from the Q-value
  |   table.
5 |   Action a takes the system to a state s' and receives a reward r.
6 |   Update Q(s, a) according to r.
7 |   s ⟵ s'
```

The learning algorithm has to try out all possible alternatives infinitely often in order to converge to the optimal policy. This involves both exploration and exploitation [38]. However, maintaining the balance between exploration and exploitation is a challenging task in reinforcement learning. Exploitation means taking an action that seems best according to the already

acquired information, whereas exploration means trying out new things in the hope of finding better actions. In order to maximize the reward, the agent has to exploit and choose the best action it knows so far. However, this does not guarantee the optimality of the solution in the long run because the chosen actions may not be the best ones, and there can be other unexplored actions which can give better long-term reward [44, 45]. Therefore, exploration is important where the agent takes a risk to try out new things with the objective of learning. There is always a general conflict between learning and obtaining high rewards. For this reason, initially there is a low weight on obtaining high rewards, and this weight is increased later when the best actions are known better. That means, in general, one should learn for a while by exploring different actions, and after certain stage one can exploit the information that was learned.

There are many approaches defined to balance between exploitation and exploration. One straightforward and successful approach is ϵ-greedy [47]. In this method, the parameter ϵ determines the randomness in action selections and controls the amount of exploration [45]. Usually the range of ϵ is between 0 and 1. The agent chooses the best action with probability 1- ϵ, and otherwise, it chooses the actions at random. There are other exploration methods that use counters [43], confidence bounds [19, 20], model learning [9], soft-max policies [31]. Among them, ϵ-greedy approach is a widely used method and one of the reasons behind it is because memorization of exploration specific data is not required [44, 45]. However, choosing a good value for ϵ can be a challenging task [45].

More information regarding exploration and exploitation schemes can be found in [42].

3.4 Types of Reinforcement Learning

Two types of reinforcement learning can be distinguished based on the number of agents *single-agent learning* and *multi-agent learning* [31]. Single-agent learning is a very simple form of learning where there is a presence of only one agent and the environment remains stationary. The agent optimizes its behavior by choosing actions that maximize the reward. The idea behind multi-agent learning is the same as single-agent except that there are several agents interacting with the environment as well as communicating with each other.

Learning in the multi-agent framework is more difficult than the single-agent environment. Unlike in single-agent learning, the reward in multi-agent learning is based on the combined actions of agents. So, choosing the best

joint action for obtaining the best reward is difficult and it takes a long time for learning to converge.

Consider an example from [11] shown in Table 3.1. There are two agents and each of them has a choice of two actions. Agent 1 has actions a_0 and a_1 and Agent 2 has actions b_0 and b_1. The reward depends on the decisions made by both agents. The best rewards are obtained when agent 1 takes action a_1 and agent 2 takes action b_1. But, during learning phase, agent 1 is uncertain about choosing action a_1 because it could get better reward with a_0, if agent 2 takes action b_0 instead of b_1. Hence, learning in the multi-agent framework is difficult as each agent, very often, ignores the decision of other agents. A strong coordination among the agents need to be built to get the best rewards.

	a_0	a_1
b_0	5	0
b_1	0	10

Table 3.1: A two-player game. Actions a_0 and a_1 belong to Agent 1 and actions b_0 and b_1 belong to Agent 2.

There are two forms of multi-agent learning based on how the agent interacts. They are independent (centralized) learning and joint (distributed) learning [11, 48].

Independent Learning Agents in independent learning are unaware of the existence of other agents or they ignore the impact of the other agents' behavior on their rewards [11]. Thus, learning for each agent is same as single-agent learning. Each agent chooses its own actions and optimizes its behavior based on the rewards obtained from those actions.

Joint Learning Unlike in independent learning, each agent in joint learning cares about the presence of other agents and shares information among each other, and receives a joint reward from joint actions. In other words, the focus of agents is on maximizing joint reward instead of maximizing their individual reward.

Tan [41] suggested three different ways of agent cooperation in multi-agent RL and presented case studies for each of these approaches. In the first approach, agents can communicate with each other by sharing immediate information such as sensation and rewards. In the second, agents share the

past experiences (that others may not have experienced yet) in the form of episodes which is the triples consisting of sequence of a state, an action, and a reward. In the third, agents share learned policies. The results show that cooperative learning is faster and the convergence is quicker compared to independent learning. However, state spaces in cooperative learning are very large and the communication costs are high. Therefore, there is a trade-off between the performance and cost when cooperative learning is used.

Weiss [48] studied coordination among agents in multi-agent learning and proposed learning algorithms that make the agents act concurrently. He studied reinforcement learning in a setting where only one agent gets to perform and action, and all agents have to compete to get the right to be that agent.

Wiering [49] proposed a multi-agent model-based RL method for a traffic light control to optimize the driving policies of cars. He considered three different RL designs for using current information about the state of other traffic lights. In the first one, only local information is used to compute waiting time of each car. The second one used global information for computing waiting times for the first car in the queue and local information for other cars. The third one used global information for all cars. The RL designs were compared against non-adaptable traffic light controller and the results showed that the RL designs perform better. Furthermore, the performance of each design depends on the number of cars added at each time step.

Lauer and Riedmiller [26] used a model in which multiple agents learn a policy independently of other agents, all trying to maximize a shared reward function. This work has the interesting aspect that, similarly to the cellular network management problem, the agents' goal is to maximize the overall joint reward at the cost of their own private reward (which are a component of the joint reward).

3.5 Reinforcement Learning in Network Management

Reinforcement learning and related techniques have earlier been used in the management of mobile networks. Razavi et al. [35] combined reinforcement learning with fuzzy logic to control antenna tilt in LTE networks. The control of a base station antenna is represented as rules with a fuzzy condition. The state space is abstracted according to qualitative parameter values, and reinforcement learning associates the best possible action with each rule. In this work, a global (rather than a distributed/decentralized) policy is learned,

and the feasibility of the approach is demonstrated with a model that has 7 base stations with 3 antennas for each.

Bennis et al. [7] applied decentralized RL to channel selection and power control of networks of micro, femto, and pico cells. The algorithm jointly estimates the long-term utilities of femtocells and optimizes their strategies.

Bennis and Niyato [6] proposed a distributed Q-learning algorithm for managing interference between macrocells and femtocells as well as among neighboring femtocells. The algorithm was effective in self-organization of femtocells where each femtocell learns from the local environment and was able to make decisions on their own. Moreover, the learning algorithm was able to reduce interference towards the macrocell as well as manage interference among the femtocells.

Bernardo et al. [8] studied the feasibility of RL-based framework for Dynamic Spectrum Management (DSM). The proposed framework was successful in performing spectrum assignment per cell dynamically in radio access networks. In addition, RL algorithm stands out the best to balance between spectral efficiency and QoS compared to other strategies of spectrum assignment.

Lee et al. [27] proposed a Q-learning based approach for autonomic reconfiguration of wireless mesh networks with the objective of improving the network performance. Each network node in the system was able to learn a good policy in the changing environment and improve the quality of packet delivery ratio.

Other research on network management using RL can be found in [15, 28, 32].

Chapter 4

Learning Algorithm for Adaptive Networks

The mobile network management problem, deciding about the configuration of antenna parameters such as transmission power and antenna tilt, can be viewed as a reinforcement learning problem in which the quality-of-service of mobile devices, in the range of the base station, is viewed as the reinforcement signal. There are many configuration parameters, and in this work we have chosen parameters relevant to CCO SON use case. The main reason behind using the RL approach is that, in this problem, we do not know the reward associated with each action in advance and hence cannot decide immediately the best settings for each cell. Additionally, the situation in the network is dynamic and one-time optimization of the system is not possible even in principle, rather one needs to optimize continuously as situation evolves. Therefore, we need to try out various possibilities of changing the network parameters. The decisions on choosing the best action are gradually improved by utilizing the feedback obtained from earlier decisions. The feedback in the network management can be the QoS measured in terms of better connection quality, for example. However, this application —network management— has several features that differ from the most commonly used reinforcement learning methods.

Distributed RL The size of joint state space is N^M where N is the number of states per base station and M is the number of base stations. Due to this large state space, the use of policies that map the joint state space of all base stations to a control action is infeasible. Distributed and multi-agent variants of reinforcement learning [48] seem a better match for the network management application rather than doing single-agent RL with the whole network.

State-space structure The state space with respect to the effects of possible control actions is simple: for the current state of the system (i.e., the mobile terminals and their usage status with respect to each base station), the antenna configuration (i.e., transmission power and tilt) can be changed from any control state to any other in one step. Thus one of the main difficulties in reinforcement learning in general, the potentially very long action sequences needed to get from one state to another, leading to slow learning of optimal policies, is not present in this application.

Reward function vs. observations The reward function coincides with the observation function. The management objective is to maximize the QoS measures, and the QoS measures are what is observed when making management decisions.

We consider a basic Q-learning style algorithm adapted to the specific features of the network management problem.

4.1 The Learning Algorithm

The learning algorithm for adjusting the network parameters consists of state space, abstraction function, scoring function and exploration/exploitation policy. Each of these is explained thoroughly in this chapter.

We have a high number of base stations and the learning algorithm optimizes the network parameters of each of these independently. Each antenna of a base station, corresponding to a cell, has its own parameters to adjust, and we control the Transmission Power (TXP) of each antenna so that the measure for overall connection quality improves. However, too high TXP causes interference between base stations, and too low TXP weakens connection to mobile terminals. Therefore, the algorithm has to find an optimal solution to balance between the interference and connection quality. The algorithm starts from some initial TXP settings which are defined by the network operator. The network generates a state vector called CQI, that summarizes the connection quality of the mobile terminals in the range of the base stations. This vector is already a rough abstraction of the reality of the network. However, identifying these vectors with the states for our learning algorithm will result in a very large state space. Therefore, we apply an abstraction method which maps the values in the CQI vector to fewer values. The final state for each cell is then given by the current TXP setting and an abstracted CQI vector. The algorithm then explores a new action with some randomization, or in other words, it changes the TXP values for each

cell and waits for the results from the network based on the changes. A new state is reached due to the effect of the chosen setting. We then compute a score for this new state from the CQI vector. Finally, the Q-value table for current state-action pair is updated using the score of the new state. The process continues and repeats the same procedure indefinitely. In the simulated environment, the process can be stopped after a certain number of iterations. However, one needs to run the algorithm for many iterations so that the learning converges.

Algorithm 2 presents the multi-agent and independent learning algorithm, with subprocedures being explained later in more detail.

Algorithm 2: Learning Algorithm

```
1  counter = 1
2  while True do
3      for cell do
4          s = getstate(cell)
5          %Explore action (TXP changes)%
6          if randomnumber < explorationrate(counter) then
7              action[cell] = arg max_{a∈actions} Q^{cell}[s, a]
8          else
9              action[cell] = randomly chosen action.
10     %The network continues to run with new TXP settings
       and produces new CQI vectors%
11     for cell do
12         s' = getstate(cell)
13         reward = score(s'.CQI)
14         Q^{cell}[s, action[cell]] = γ × Q^{cell}[s, action[cell]] + (1 − γ) × reward
15     counter = counter + 1
```

4.2 State Space

The current TXP value and connection quality of the terminals (based on the TXP and the physical state of the surroundings of the base station) are the relevant features of the network that define its state. The CQI vector, which is obtained frequently from network, gives the quality measure for all the terminals. Hence, we initially define the unabstracted states of the system which is composed of the TXP value and the CQI vector and later abstract these states. We term these TXP values as the *control state of the*

system. Since we know the terminals' situation in every cell of the network, we compute the state of each of these cells independently.

The size of the Q-value table depends on the size of state space. Our states are the composition of TXP values and CQI vectors. Consider 5 control states, e.g. five different levels of transmission power, and 100 different values in CQI vector. Assume that the size of a CQI vector is 15. Then, the size of state space will be 5×100^{15}. This gives the size of Q-value table as $5 \times 100^{15} \times 5$, since we need to store values for each state-action pair. The size of both state space and Q-value table is huge in real networks as they have more transmission values and different values in the CQI vector. Hence, in order to keep the size of Q-value table manageable, we need to define a way of abstracting the state.

4.3 Abstraction of the State Space

In reinforcement learning, the environment is described as a set of states and the agent learns to act optimally with the unknown environment. However, learning a good policy becomes difficult when the number of states grows exponentially. The increase in the problem parameters requires more memory and computation time [37]. Therefore, it can be necessary to abstract the environment states in order to reduce the state space size and speed up learning.

The problem of large state spaces is very common in many application domains and many methods have emerged to reduce the size of state space. There are some methods which consider the subset of state components from a set of candidate abstractions as the best abstraction [14]. In such methods, the size of abstracted state space is exponentially smaller than the unabstracted state space [14]. Seijen et al. [37] have proposed a dynamic way of abstracting the states where the agent switches among the different abstractions during the learning in order to define the state accurately. Wiering [49] has also used some abstraction to reduce the states in the traffic light control application. Razavi et al. [35] abstract the state space according to the qualitative parameter values (e.g. "low" and "high"). In some applications, state abstraction also means eliminating some of the irrelevant features in an optimal way [1].

In our case, the states are defined by the composition of the TXP value and the CQI vector. The CQI vector is a collection of classes and there are values in these classes which indicate the terminals' connection quality. The values in the lower CQI classes indicate terminals with poor connection whereas the values in the higher classes indicate terminals with good con-

nection. The number of such CQI classes, holding the information about the terminals, is large and considering all the classes leads to very large state spaces which slows down the learning process. Therefore, there is a need for some kind of abstraction in this application as well. We define an abstraction function which will use values of all the CQI classes and map them to a small number of values.

We divide the whole abstraction process into two steps. The first step involves merging the CQI classes into a smaller number of components, and in the second step, we apply the abstraction function that will map the values of each component to a single abstracted value. Below we explain both these steps in detail.

4.3.1 Partitioning the CQI Classes into Components

The CQI vector is composed of CQI bins/classes and a particular CQI class represents the number of terminals belonging to the CQI class within the measurement period. We first partition this vector into components. The division of CQI classes into a fixed number of components is based on the hierarchy of these CQI classes. As mentioned above, the lower columns/classes represent poor connection and as the classes go up the connection quality improves. Initially, we divide the CQI vector into three components and later will also consider more components. The three components can be termed as *poorconnection*, *goodconnection* and *excellentconnection*. Deciding the boundary for the component classification is one of the important tasks here. Therefore, we experimented by changing the boundaries of components and later fix the one that results in better performance. Initially, we divided the CQI classes into equal size. However, the movement of terminals from good classes to excellent classes was not clearly visible because the boundary was too far to the right. Thus, later we changed the border among them and kept fewer classes in the bad component by changing the boundary and kept more classes in the excellent component. The justification for choosing such boundaries is that it better splits the set of terminals, and the scoring of network states looks at how many terminals are *Good* and how many are *Excellent*. Essentially, an implicit objective is to move terminals from *Bad* and *Good* to *Excellent*, and if the boundary is too far to the right, it is only possible to move very few terminals, and if almost all terminals are good all the time, their moving to the right is not visible to the learning method.

In addition, we did the experiments with more components (four and eight) and compared their effect on the performance of the learning algorithm.

4.3.2 Abstracting Values for the Components

The components obtained from the first phase are then mapped to a small number of values, for example in the range from 0 to 3. We will be experimenting with different ways of abstracting the CQI vectors and see their impact on learning. The important question here is how the components are mapped to one specific value.

In order to distinguish among several abstracted values, we define a threshold value for each component that will separate one abstracted value from another. Initially, we make a guess to define the threshold values. If over 60 percent of the terminals of the base station belong to a component, the value for the component is 3. Between 40 and 60, the value is 2, between 20 and 40 it is 1, and below 20 it is 0. We run some simulations based on these guesses. Further, we analyze the results from these simulations in order to know the range of the percentage of terminals connected to each component. Algorithm 3 explains the process of deriving the threshold values.

Algorithm 3: Deriving threshold values

```
1  Input : m and n are integers such that m ≥ 1 and n ≥ 2
2  Initialize C[i] which is a multi set that contains values for each
   component i.
3  C[i] = {}
4  for each CQI vector encountered so far do
5  |  Divide the CQI vector into m components.
6  |  Compute the percentage P of terminals connected to each
   |  component i.
7  |  C[i] = C[i] ∪ {P}
8  end
9  Sort values in C[i] in ascending order.
10 Divide the values in C[i] into n parts. This will give a range of values
   for each abstracted value and the boundaries of these range of values
   are used as threshold to distinguish among abstracted values.
```

Derivation Here is a formulation of deriving the threshold that will make the things clearer. Consider, we have some k simulation steps from random assignment of threshold values. Assume, we now want to perform a new test with m components and n abstracted values. So, we use these previous k simulation results to define the threshold values in our new experiments. We compute the percentage of terminals in each component per

simulation and sort them in ascending order. That will give the vectors as follows: $c_1 = [x_1, x_2, ..., x_k]$, $c_2 = [y_1, y_2, ..., y_k]$,...., $c_m = [z_1, z_2, ..., z_k]$. Now, the values in each component are evenly split into number of abstracted values. Here, we split into n parts. Since the division is equal, the size for each abstracted value will be $p = \lceil k/n \rceil$. So, for c_1, we obtained the values for each abstracted value as follows. $absvalue_1 = [x_1, x_2, ..., x_p]$, $absvalue_2 = [x_{p+1}, x_{p+2}, ..., x_{2\times p}]$,....., $absvalue_n = [x_{n\times(p+1)}, x_{n\times(p+2)}, ..., x_k]$. Division for other components is made in the same way. Now, the range of values belonging to an abstracted value will define the threshold. So, in this scenario, for c_1, if the percentage of terminals is less than x_p, then c_1 is mapped to $absvalue_1$. Similarly, if the percentage of terminals is greater than or equal to x_p and less than $x_{2\times p}$, then it is mapped to $absvalue_2$ and so on for other cases as well. Table 4.1 illustrates the derivation where we have three components and each component can be mapped to three different abstracted values.

Hence, in this way, we obtained threshold values for each component and this will determine the abstracted values. This method of threshold derivation is appropriate because from the past simulations we can learn the average distribution of terminals in each class and hence this will help to abstract network state in new simulations. One can frequently update these threshold values by learning more from past simulations as well.

	Component 1	**Component 2**	**Component 3**
Value 1	0.15, 0.1561,.....	0.46, 0.477,.....	0.28, 0.29.....
Value 2	0.17, 0.1753,.....	0.496, 0.5004,.....	0.32, 0.329,.....
Value 3	0.18,0.195,.....	0.5121, 0.52,.....	0.35, 0.359,.....

Table 4.1: Threshold value derivation

We have explained the process of abstraction function in detail and now will give an example. Consider 16 CQI classes and let the observation time period be t. Assume (v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}) is a vector for a cell obtained by observing CQI values. Now, these vectors are divided into three components, for example of equal size. That means, the values v_0, v_1, v_2, v_3, v_4 represent the first component where terminals have a poor connection, v_4, v_5, v_6, v_7, v_8, v_9 represent the second component where terminals have a good connection, and remaining vectors

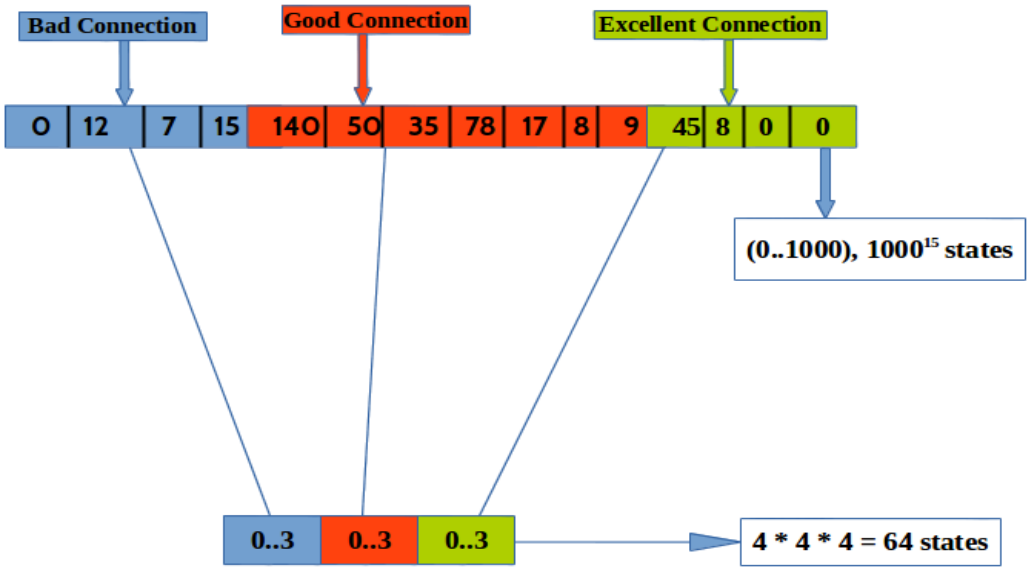

Figure 4.1: Abstracting Connection State

represent the third component where terminals have an excellent connection. Each of these components is then mapped to a value in the range of 0 to 3. The combination of these mapped values will then give the abstracted state.

Figure 4.1 illustrates the abstraction process. In this diagram, the CQI vector is divided into three components and each of these components is mapped to one of the four values. Abstraction has reduced the number of states: considering 4 values, the abstraction will result in only 64 states, whereas without abstraction there are 1000^{15} states. In the figure, we assume there are 15 CQI classes and each class can have 1000 different values, then each different value in a class indicates a unique state.

The complete state space is the combination of the *control state of the system* and the *abstracted CQI vector*. In the above example, if we consider 20.0, 24.0, 30.0, 27.0 as possible TXP values and 0, 1, and 3 represents the abstracted values then $(20.0, [0, 1, 1])$, $(24.0, [0, 1, 3])$ are the states of the system.

The level of abstraction of the CQI vector can be either high (coarse) or low (fine). Coarse abstraction means either the CQI classes are partitioned into a smaller number of components or the components are mapped to few abstracted values. On the contrary, fine abstraction means the CQI classes are partitioned into more components or the components are mapped to more values. As a result, fine abstraction will increase the combination of the abstracted values, thereby increases the size of state space. Hence, there

is a trade-off between the level of abstraction and quality of control: Fine abstraction allows more fine-grained distinctions between physical states, and hence better choice of control actions. However, it also increases the number of states that need to be considered, which slows both computations and the rate of learning.

4.4 Actions

The actions represent the changes in the control state. After the control state is changed, a new state can be observed. For example, if the current state is $(20.0, [val_1, val_2, val_2])$, then increasing the transmission power by 2.0 can be one of the actions that leads to a new state, $(22.0, [val_1, val_3, val_2])$.

4.5 Learning Rate

In this section, we discuss the exploration and exploitation policy used in our learning algorithm. In mobile network management, our main objective is to improve the connection quality of mobile terminals and this is known from the CQI vector. Hence, the objective here is to move the terminals from lower CQI classes to higher classes. In reinforcement learning, we have the choice between taking the best action (according to current information) and maximizing future rewards, and learning more by taking an action with a lower value. Thus, we define an exploration rate in such a way that the algorithm will have a chance to explore more in the beginning and later it can focus on maximizing the reward. Hence, according to this exploration function, the probability of choosing the best action increases gradually over time. To be able to adapt to a changing network environment, other actions than the best one have to be tried occasionally as well. We experimented with two different exploration functions based on the degree of randomization.

1. In the first function, the probability of choosing the best action outright is zero in the beginning and later it increases. Equation 4.1 is one of the ways of defining the exploration function.

$$prob(n) = 0.95 - 0.95 \times 0.7^{n/40.0} \tag{4.1}$$

   Here, n is the counter for time step. This function yields $prob(0) = 0$, $prob(100) \approx 0.5$ and $p(300) \approx 0.9$.

2. In the second case, we randomize less so that the emphasis is given to the best actions most of the time.

$$prob(n) = 0.97 - 0.30 \times 0.8^{n/30.0} \tag{4.2}$$

This function yields $prob(0) = 0.67$, $prob(100) \approx 0.8$ and $p(300) \approx 0.9$.

Figure 4.2 depicts these two functions. With the second function, most of the cells remain with initial/best settings and experimentation is done with fewer cells at a time. Learning is faster with more randomization whereas less randomization is conservative and slow in adapting the changing environment. However, the system behavior will be more stable when learning is performed with less randomization.

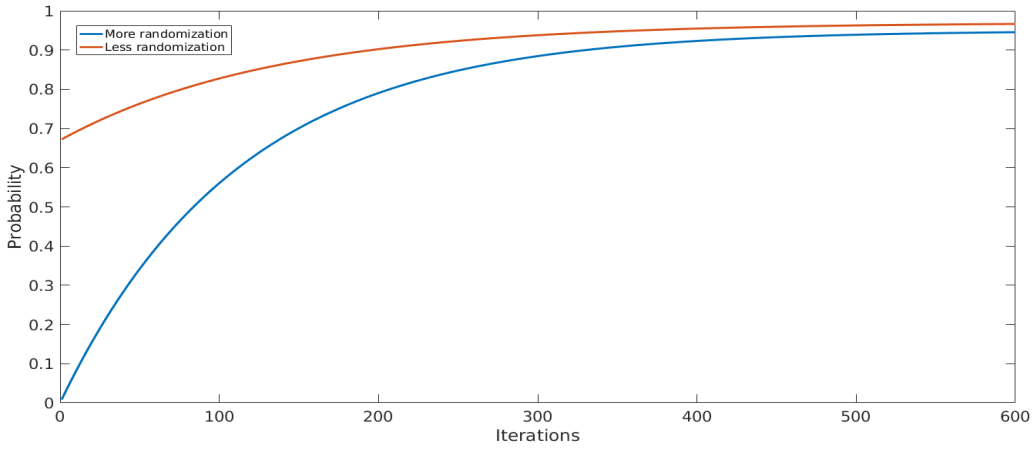

Figure 4.2: Exploration Rate: The curve shows the probability of choosing the best actions at each time step.

We also tried other exploration strategies in the beginning of the work, which did not work out well.

1. When choosing an action for state s, we add a random number to the Q-value of state-action pairs, i.e., Q(s,a), and then select the one that has the maximum value.

   Let S be a set of states and A be a set of actions. Assume, for any state $s \in S$, action $a_1 \in A$ gives the maximum Q-value and action $a_n \in A$ gives the minimum Q-value. Then, for each action $a \in A$, associate random number is

$$R_a = (Q(s, a_1) - Q(s, a_n)) \times random(0, 1). \tag{4.3}$$

Here, $random(0, 1)$ returns a real number from the range 0 and 1. Then, the best action $a \in A$ for state s is the one that maximizes $R_a + Q(s, a)$, i.e.,

$$argmax_{a \in A}(Q(s, a) + R_a). \tag{4.4}$$

2. Actions that currently look better are chosen with a higher probability. Algorithm 4 formalizes this method.

Algorithm 4: Exploration

```
r = random()
if r < thresholdvalue1 then
    Choose best action
else if thresholdvalue1 ≤ r ≤ thresholdvalue2 then
    Choose second best action
else
    Choose third best action
```

4.6 Scoring Function for States and Abstract States

In order to know how good or bad an action is, we need to map the feedback given by the environment to real values. This mapping is defined by a function which we call a *scoring function*. The scoring function computes a score for each state where the state is only a CQI vector and it does not include the control part. We design the function in such a way that if a successor state has a better quality of service then the score is higher. Improving the connection quality of the terminals is more important in lower classes. This is because the terminals in higher classes are already in a good state. Hence the score function is designed accordingly: Moving a terminal from a poor connection to a good connection improves the score more than moving from good to excellent.

We present two different methods for computing the score.

4.6.1 Unabstracted Score

In this case, the scoring function uses the unabstracted CQI vector where the values of each CQI class are considered separately. We first define a weight

vector such that weights are assigned to all the CQI classes. Terminals in lower classes are more important to deal with than terminals in higher classes. Hence, weight difference between consecutive lower classes are kept bigger than between consecutive middle classes, and all higher classes have the same weight. Figure 4.3 shows the weight vector for computing the unabstracted score. The x-axis is the CQI classes and values in y-axis indicates the weights to each CQI class.

Then the score is computed as the dot product of the value in the vectors with given weights of each CQI class. The steps are as follows.

1. Define weights for each CQI class.
   For example, $W = (-10, -8, -6, -4, -2, 0, 1, 2, 3, 4, 5, 5, 5, 5, 5, 5)$

2. Compute the percentage of terminals in each CQI class of a state $s = (v_1, v_2, ..., v_n)$.

$$x_i = \frac{v_i}{\sum_{i=1}^{n} v_i} \tag{4.5}$$

3. Compute the sum of the products of the percentages of terminals with weights of each CQI class.

$$score_u(s) = \sum_{i=1}^{n} w_i x_i \tag{4.6}$$

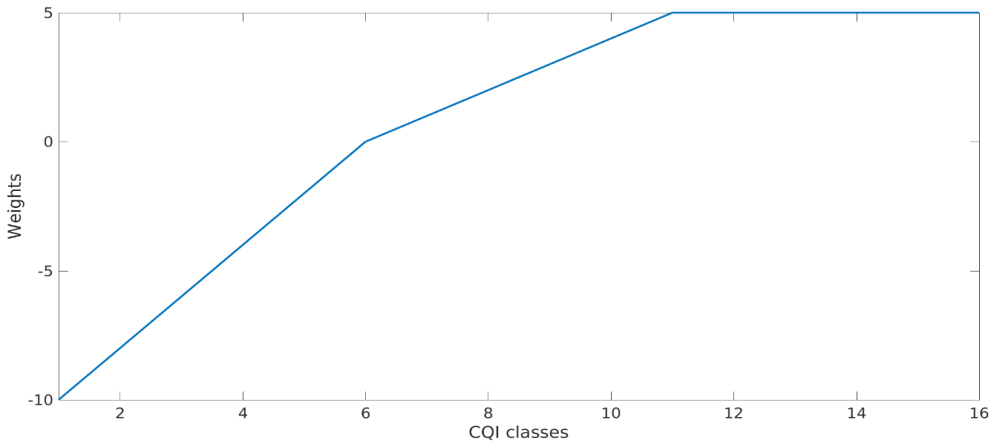

Figure 4.3: Weight vector for CQI vectors

4.6.2 Abstracted Score

We also define the score on the basis of abstraction mentioned in Section 4.3. The reason behind using this approach is that it enables to handle all state abstraction at one point in the interface between learning algorithm and the network. Since, the abstracted states are defined from the CQI vector, score can be given directly to abstracted states rather than individual CQI classes which prevents multi-processing of the CQI vector. Here, we first define a weight for each CQI component instead of all the CQI classes as in the unabstracted case. Next, the abstracted value of each component is mapped to a real value. Finally, we sum the product of weights with the mapped value.

Initially, we chose a linear weight vector to compute the abstracted score. Later, we derived weights as the mean of weights from the unabstracted weight vector which gives the weight vector in piecewise linear form. For example, if the number of components is three, then the weight of the first component is the average value of weights of classes belonging to the first component. We experimented with both linear and non-linear weight vectors and the results of comparisons can be found in the experimental part of this thesis. Figure 4.4 shows the two types of weight vectors used for computing the abstracted score. Here, we assume that there are eight components in total and the numbers in y-axis represent the weights of each component.

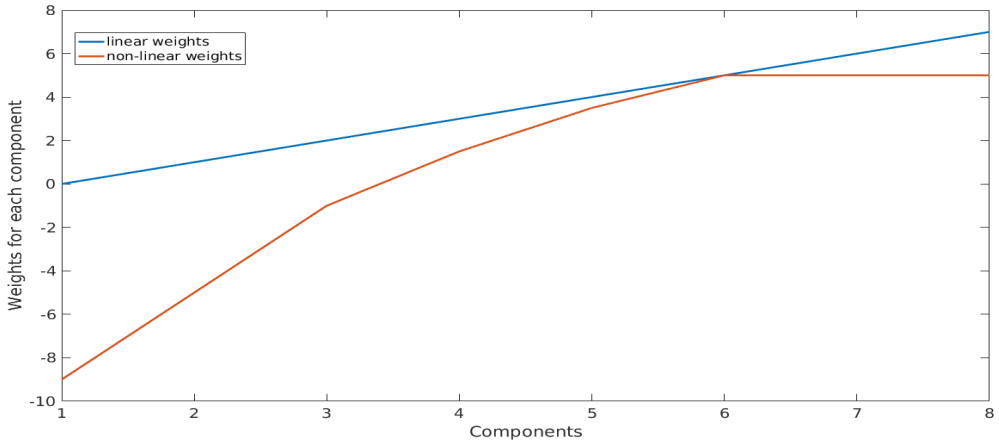

Figure 4.4: Weight vectors for abstracted states

Equations 4.7 and 4.8 represent the two different scoring functions with linear and non-linear weights respectively. Here, W_1 is a vector of linear weights and W_2 is a vector of non-linear weights. The lengths of both of these vectors are equal to the number of components. Here, s is a CQI

vector and $abstract(s)$ returns real numbers for abstracted values of the CQI vector.

$$score_{a_1}(s) = W_1 \cdot abstract(s) \tag{4.7}$$

$$score_{a_2}(s) = W_2 \cdot abstract(s) \tag{4.8}$$

Example: Assume the CQI classes are divided into three components, *poorConnection*, *goodConnection*, and *excConnection* and each of these components is mapped to values val_1, val_2, and val_3. Assume the weights are assigned in a linear fashion to the components and if val_1 is mapped to 0, val_2 to 1, and val_3 to 2, then the scoring function in this scenario is,
$score = 1 \times poorConnection + 2 \times goodConnection + 3 \times excConnection$
$= 1 \times val_1 + 2 \times val_2 + 3 \times val_3 = 1 \times 0 + 2 \times 1 + 3 \times 2 = 8$.

Mapping from abstracted value The abstracted value is mapped to a real number which is then used in the scoring function to compute the score. In our experiments, we initially mapped these abstracted values in a linear fashion in all the components. Later, we used the proportion of terminals falling in the range of each abstracted value. So, the same abstracted value, for example, val_1 in the bad component is different than the val_1 in the excellent component and mapping to same real value may not provide a better estimation of score. To derive the appropriate value we follow the process of threshold derivation explained in Subsection 4.3.2 where we have a range of real values for each abstracted value and the median of the range of the values is used as the mapping value for score computation. We carried out the experiments both with linear mapped values and median values and comparisons will be carried out in the experimental section.

4.7 Rewards for Actions

Given an action a that takes a state s to a new state s', the reward for the action a can either be the score of new state s' or the difference between the score of successor state s' and current state s. We used both unabstracted and abstracted scoring functions to compute the reward for actions.

Equation 4.9 defines the reward computed using the score of unabstracted state. In this case, the reward is the score of new state s'.

$$R_u(s, s') = score_u(s') \tag{4.9}$$

Equation 4.10 defines the reward which is also obtained using unabstracted score, but in this case the reward is the difference between the score of the successor state s' and the current state s. In this case, the unabstracted score is computed for each CQI vector and the reward is the difference of scores of current CQI vector and the previous CQI vector. Hence, if the score is better in current situation than in the previous situation then the reward will be positive and otherwise it is negative.

$$R_u^d(s, s') = score_u(s') - score_u(s) \tag{4.10}$$

Further, the reward defined by Equation 4.11 is based on the abstracted score of new state s' and in this case the scoring function uses linear weight vectors.

$$R_{a_1}(s, s') = score_{a_1}(s') \tag{4.11}$$

Equation 4.12 defines another way of computing reward where the abstracted score is obtained using non-linear weights.

$$R_{a_2}(s, s') = score_{a_2}(s') \tag{4.12}$$

4.8 Q-Value Updates

The next important part of the learning algorithm is updating the Q-value table. The Q-value update for each state-action pair (s, a) is defined by Equation 4.13 where the current state is s and taking action a leads to a new state s':

$$Q'(s, a) := \gamma \times Q(s, a) + (1 - \gamma) \times R(s, s'). \tag{4.13}$$

Here, γ is a constant determining the speed of learning, i.e., the impact of new information on $Q(s, a)$ value typically in the range $0.95 < \gamma < 1$. The reward for action a when it changes the state from s to s' is $R(s, s')$, and the possible ways of computing it were just discussed in Section 4.7.

The update function is simpler than in standard Q-learning because future rewards do not need to be considered, due to the simple structure of the state space. Future rewards only depend on the respective future actions, not on the current action.

4.8.1 Choice of γ

Deciding the value of γ is another important task in the Q-learning algorithm. If γ is constant, then either learning is too slow initially or there is too much noise in the Q-values. Hence, in order to make its value dynamic, we count the number of times the state-action pair appears in the learning process and define γ as

$$\gamma = 1 - \frac{1}{min(count(s,a), 100)}. \tag{4.14}$$

Here, s is a state and a is an action. So, if a state-action pair appears for the first time, then the value of γ is 0 and later the value of γ increases. This way the Q-values initially change quickly and after several learning steps the change will be slower. Figure 4.5 shows γ as a function of $count(s,a)$.

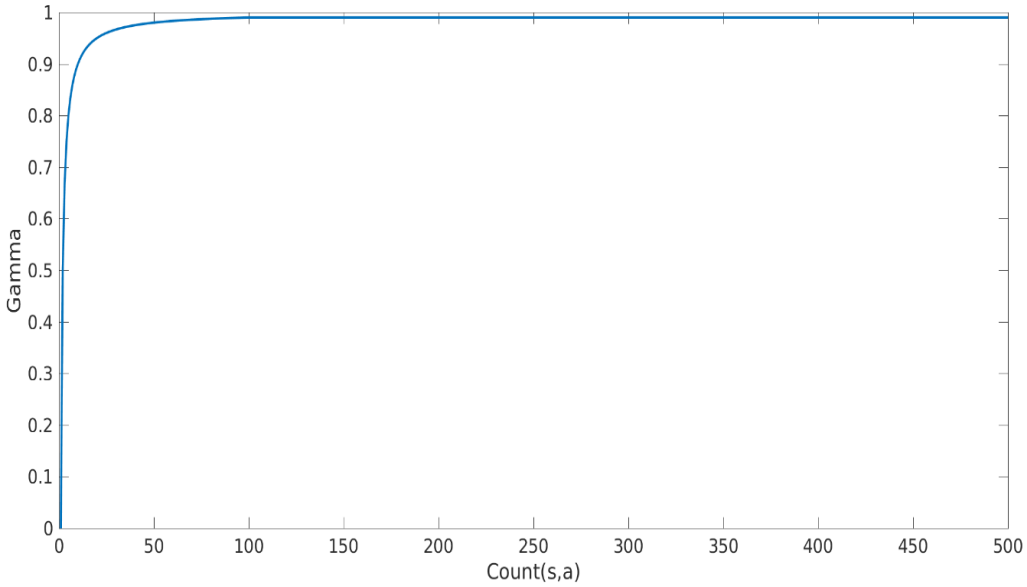

Figure 4.5: Curve showing γ

Chapter 5

Experiments and Results

In this chapter, we study the applicability of our methods by performing experiments and we try to answer the following questions.

1. How much does learning improve network performance?
2. What kind of abstraction of states is best?
3. How randomization in action selection affects learning?

5.1 Setting

We used a software simulator for mobile networks that was developed by Nokia to carry out the experiments. This simulated network consists of 12 LTE macro cell base stations and 32 cells in total. We used about 2000 terminals moving at random around the simulation area which is an urban area in central Helsinki. The network structure and positioning of the cells is shown in the screen-shot in Figure 5.1. The simulator at first runs with initial transmission settings and it produces the data such as CQI and RLF values in every 15 minutes interval of time and the parameter changes are made after each measurement round. The CQI vectors produced by the simulator had 16 classes.

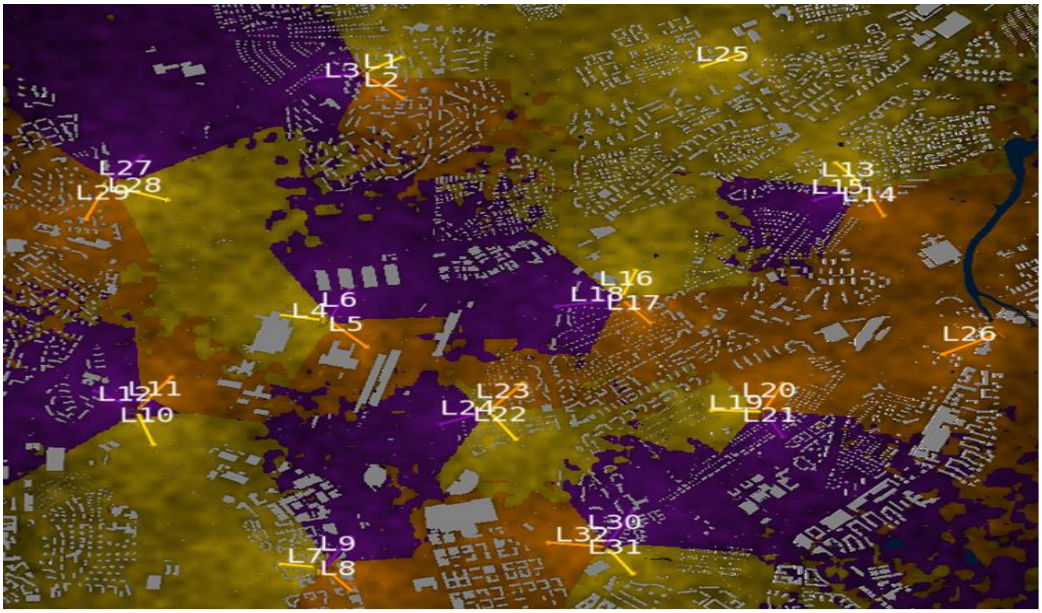

Figure 5.1: A screen-shot of the LTE network simulator

The initial TXP value for all cells is 40.0. We defined three actions, *increment TXP value*, *decrement TXP value*, and *no change*. Initially, we tried to increase and decrease the TXP by a small value, for example 1 or 2. However, the impact of such small TXP changes had negligible impact on network performance. Moreover, too small changes in the TXP values would increase the size of state space. For example, if the range of TXP value is between 0 to 40 then changing the value by 1 would generate 40 different TXP values and each is consider as a unique state. On the other hand, too big change will make the network unstable. Hence, we decided to change the value by 4 after some experimentation. This defines the action space as $\{-4, 0, 4\}$.

The minimum and maximum TXP values for each cell were also fixed to constant values 0.0 and 40.0, respectively. The switching from one TXP state to another should remain within these minimum and maximum values.

The whole learning algorithm was developed as a Python program. We had a separate class for reading and processing the data produced by the simulator, and writing the TXP changes in the format so that the simulator can read the changes. In addition, the important components of the learning algorithm such as abstraction, exploration, and Q-value update are implemented by separate procedures. The whole code is of around 1100 lines. The simulator produces the data after every 15 minutes of interval and hence, the algorithm needs to wait unless it gets a data based on new TXP changes.

Due to this, running the algorithm for further simulations takes more time.

5.2 Experiments

We performed around 500 simulations for each experiment, and produced results with all versions of the scoring function. We experimented with the following aspects of the learning algorithm.

- Different levels of abstraction
- Different ways of computing scores
- Variations in exploration policy
- Linear and non-linear weight vectors for computing scores

Basically, we classified the experiments into two categories. The first category involves the experiments performed by changing the number of abstracted values (to which each component are mapped to) and the second one involves the experiments with different number of components.

5.2.1 Base-Line Model

In our base-line model, the base stations' antenna configuration settings remain unchanged. We used TXP = 40.0 for all base stations which was suggested by Nokia as it seems reasonable settings for the base-line model. We tried with other TXP values (30.0 and 25.0) as well but the results are better with TXP value, 40.0.

5.2.2 Different Number of Abstract Values

We ran the RL-algorithm with two, three, four, six, and eight abstracted values keeping the number of components constant in all cases. We used all the three reward functions (R_u, R_u^d, and R_{a_1}) for this comparison. Figures 5.2, 5.3, and 5.4 show scores during the learning process where the reward was computed using the functions R_u, R_u^d, and R_{a_1}, respectively. In all three approaches, the score is increasing and improves on the base-line model. In R_u^d , learning with 8 abstracted values is performing well; in R_u, both 3 and 8 abstracted values look better, and in R_{a_1}, the difference is not clear. We anticipate that this is due to the noise in simulation and also partly due to the randomization in choosing the actions.

Most of the curves in all three figures initially go down for a certain period of time. This is due to the randomization used in exploration. Hence, initially the score decreases, and once the algorithm learns, it starts to select the best action with high probability. A more conservative learning strategy with less randomization would avoid this dip in the connection quality.

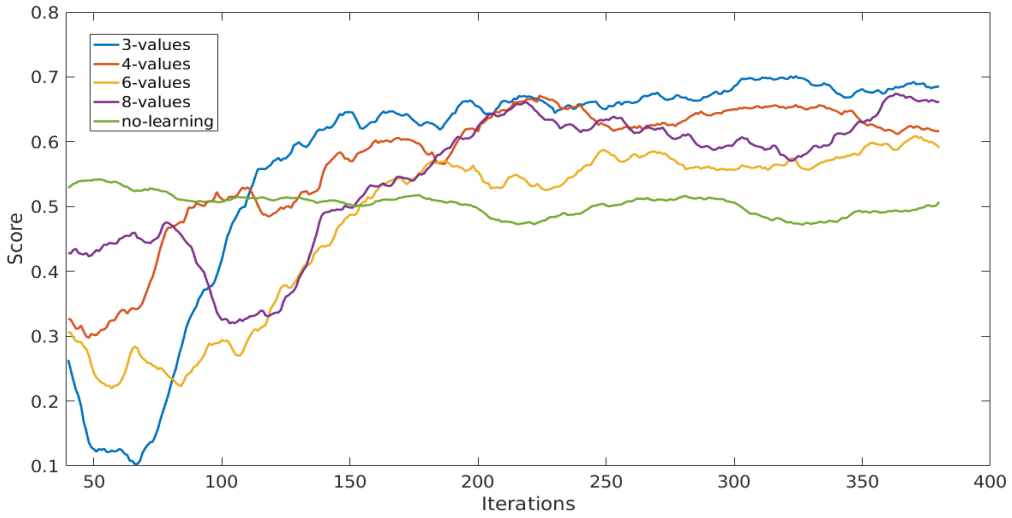

Figure 5.2: Scores based on different numbers of abstract values with reward function R_u

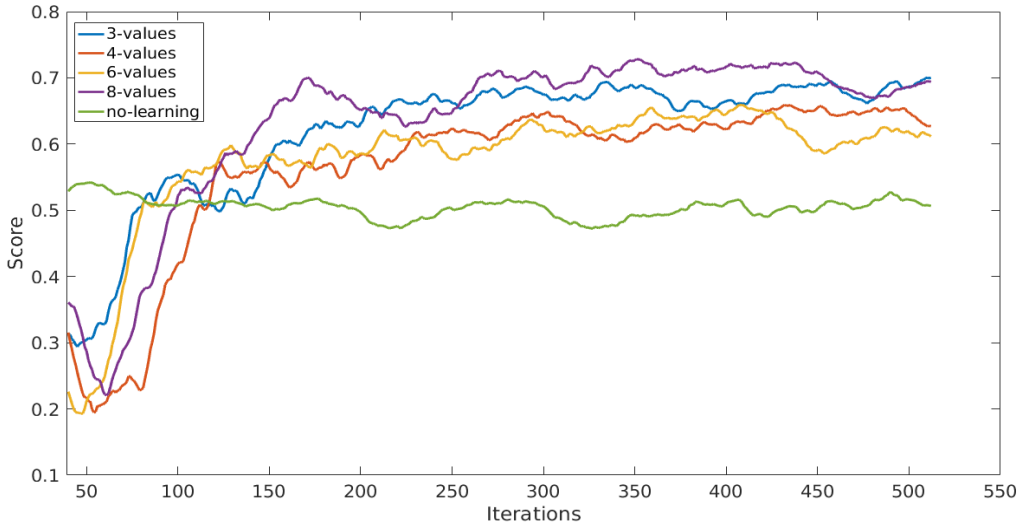

Figure 5.3: Scores based on different numbers of abstract values with reward function R_u^d

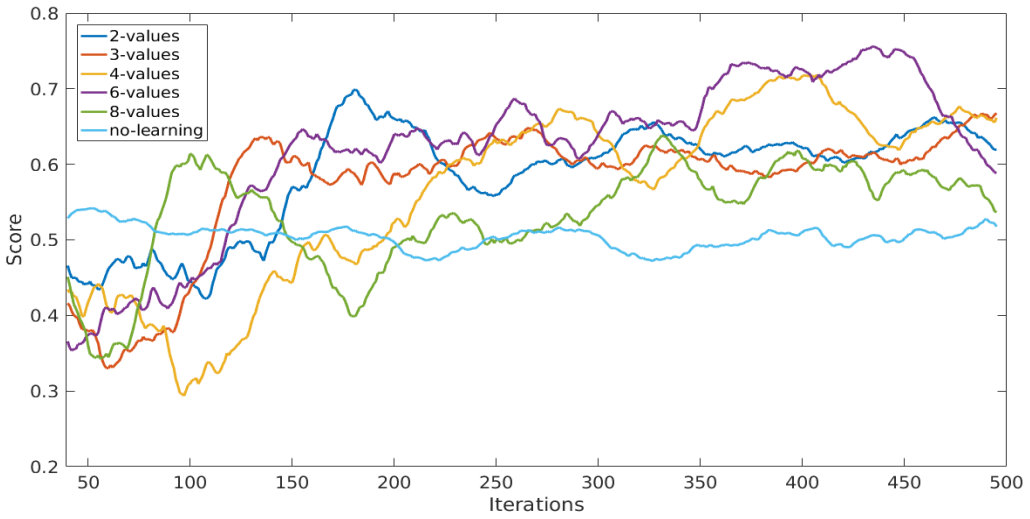

Figure 5.4: Scores based on different numbers of abstract values with reward function R_{a_1}

Figures 5.5 and 5.6 show terminals' behavior in some of the CQI classes when the learning was performed with unabstracted reward, R_u and R_u^d, respectively whereas Figure 5.7 show terminals' behavior when the learning was performed with abstracted reward R_{a_1}. Classes 2 and 3 in Figure 5.5 are lower classes with poor connection quality and there is a slight decrease in the number of terminals. On the other hand, classes 9, 10, 11, 12 are higher classes where the number of terminals is increasing. The situation is similar in other two figures as well. Thus, we can say that the terminals are moving from lower classes to higher classes thereby the improving the connection quality.

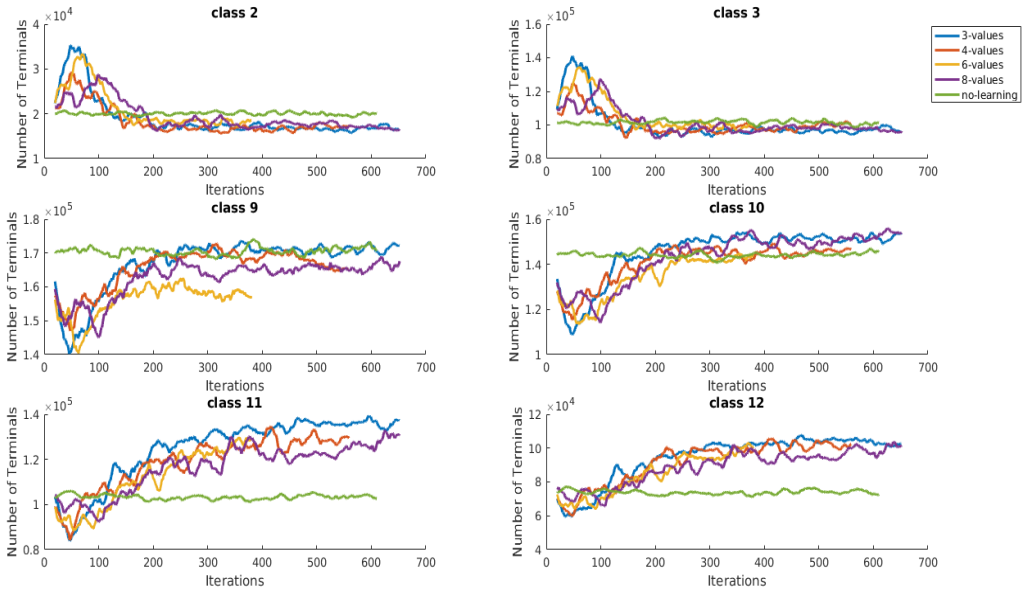

Figure 5.5: Change in terminals' behavior for a selection of CQI classes with reward function R_u

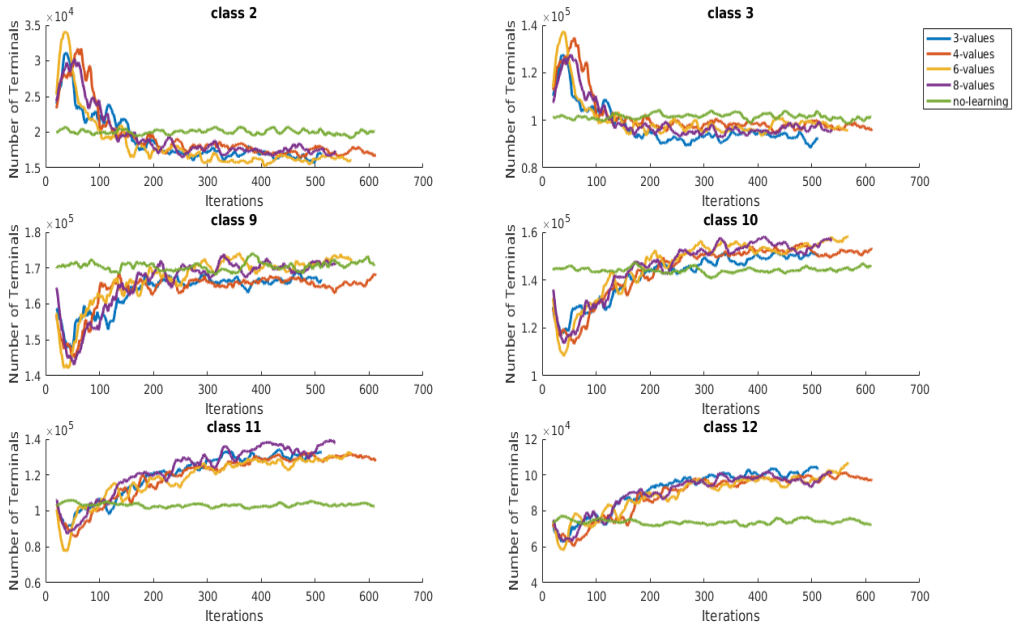

Figure 5.6: Change in terminals' behavior for a selection of CQI classes with reward function R_u^d

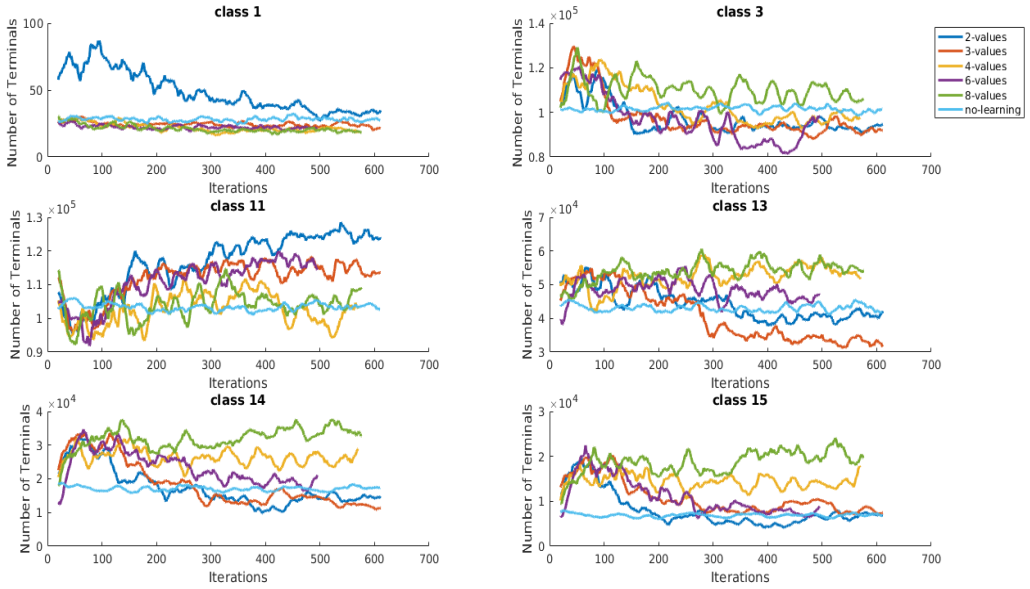

Figure 5.7: Change in terminal's behavior for a selection of CQI classes with reward function R_{a_1}

5.2.3 Different Numbers of Components

We also experimented with different numbers of components. Figure 5.8 shows the score when the abstracted states have two, three, four, and eight components.

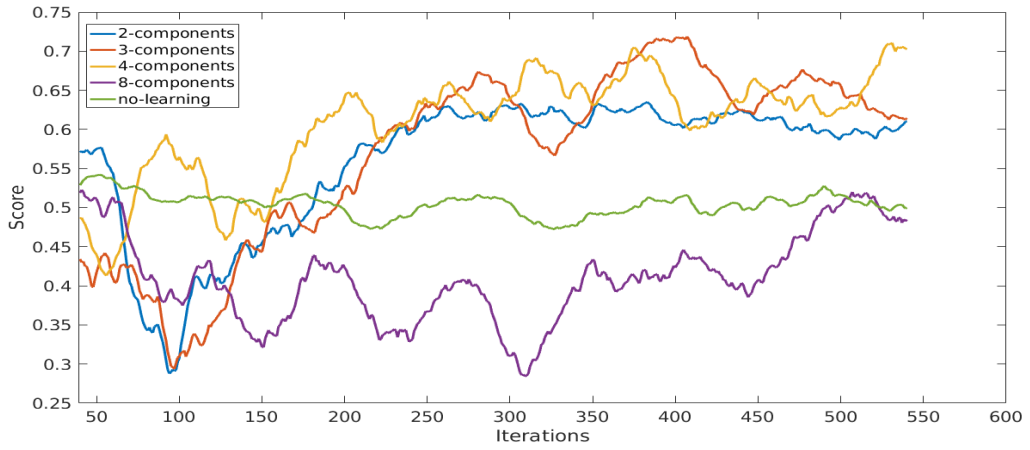

Figure 5.8: Score based on different numbers of components with reward function R_{a_1}

The score curve for 8-components is an anomaly since the scores are below the base-line model. We are not sure about the reasons for this but we think that this is due to the very large Q-value table which leads to very slow learning. Here, it is also difficult to say what number of components is the best as the scores for all of them (except 8-components) are similar.

The class-wise plots with different numbers of components are shown in Figure 5.9. As before, the numbers of terminals are decreasing in lower classes and increasing in higher classes. Learning with 2 components performs well. However, the score curve from Figure 5.8 shows that all scores based on two, three and four components yield similar improvement.

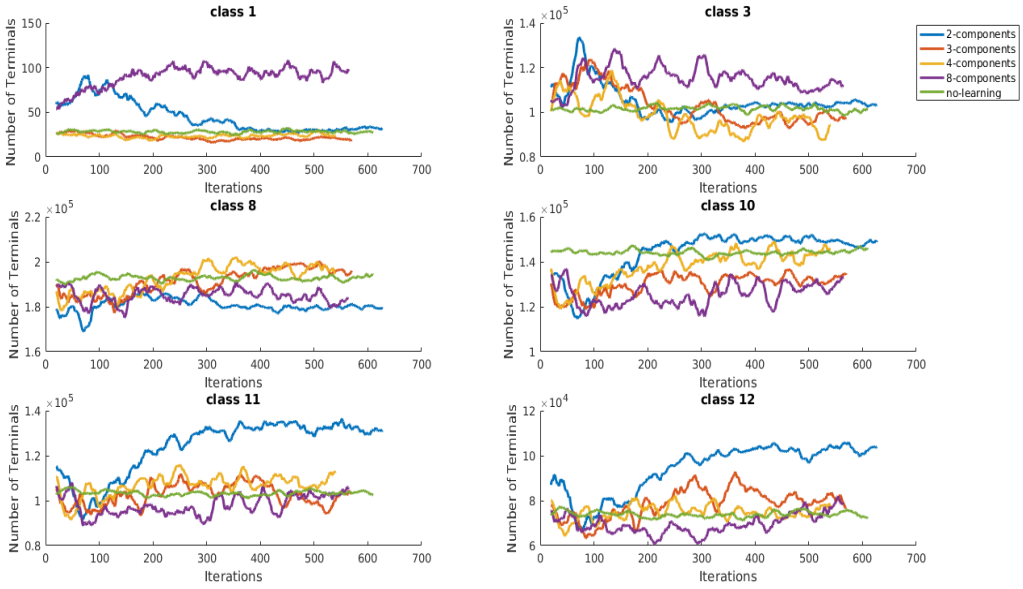

Figure 5.9: Change in terminals' behavior for a selection of CQI classes with reward function R_{a_1}. Comparisons are based on different number of components.

5.2.4 Weight Vectors and Randomization in Choosing Actions

In all the above experiments, there is more randomization in choosing the best action in the beginning. In addition, we used linear weights and linear values for computing the abstracted score. Later, we changed the weight vector and used less randomization in choosing the action and experimented with reward function R_{a_2}. In addition, we used non-linear numbers for each abstracted value while computing the score. Due to less randomization,

the algorithm takes better actions with high probability. Figure 5.10 shows the scores between the older (Version 1) and newer version (Version 2) of 8-components where version 2 has better performance .

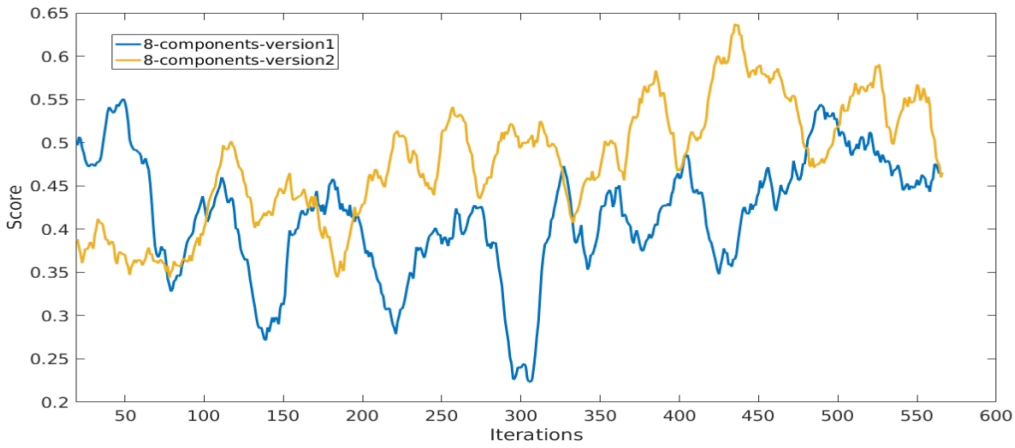

Figure 5.10: Scores for the two versions with reward functions R_{a_1} and R_{a_2} using 8 components

Figure 5.11 shows the terminals' behavior in two different versions of 8 components. The newer version is a bit smoother and better than the older version.

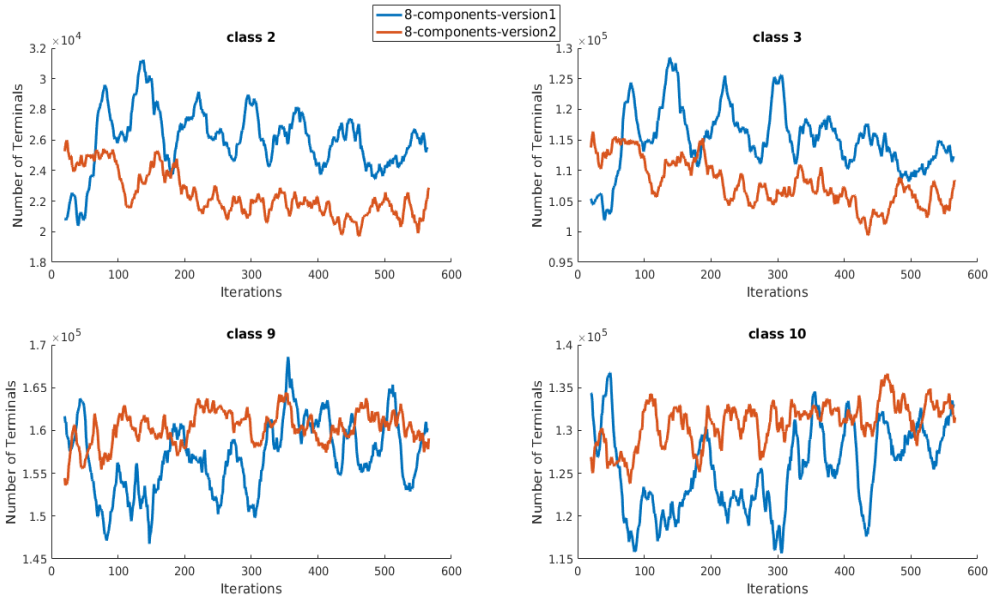

Figure 5.11: Class-wise comparisons between the two versions with reward functions R_{a_1} and R_{a_2} using 8 components

Figures 5.12 and 5.13 show scores for the two cases using 4 values and 3 values, respectively.

The main difference between these two versions can be seen in the beginning of learning. In the older version, the score curve initially goes down, but in the newer version, this is no longer the case. In Figures 5.10, 5.12, and 5.13, the scores for the newer versions start increasing from the beginning in contrast with the older version. This is due to less randomization in choosing the action. However, learning for both cases is similar afterward.

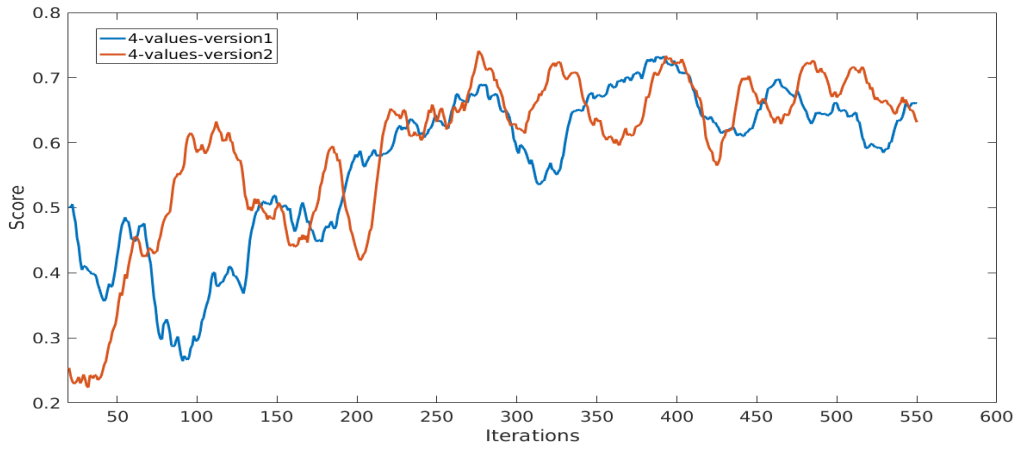

Figure 5.12: Scores for the two versions with reward functions R_{a_1} and R_{a_2} using 4 abstracted values

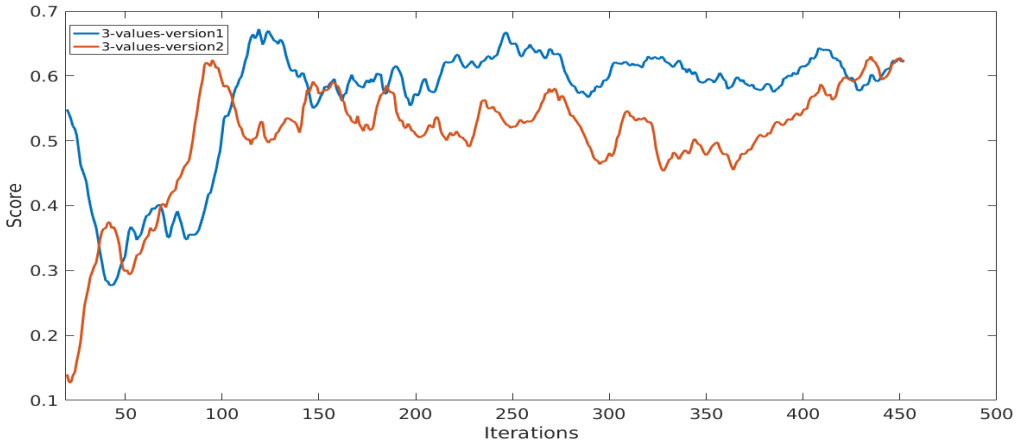

Figure 5.13: Scores for the two versions with reward functions R_{a_1} and R_{a_2} using 3 abstracted values

5.3 Result Analysis

The curves from most of the experiments are highly fluctuating. This fluctuation is due to the noise in the simulation. In addition, the noise is due to the randomization in choosing TXP settings.

Nevertheless, there is an improvement in the score function in most of our experiments which means that the algorithm is gradually finding and using better settings. Moreover, the class-wise plots also show that the terminals are moving from lower quality classes to higher quality classes. But, on the other hand, the terminals' behavior does not change much with the baseline model and the experiments show that the RL algorithm is doing better. Hence, in reference to our first question, posted in the beginning of the chapter, learning is effective and the performance of the network is improved.

Furthermore, the results with different levels of abstraction have similar improvements, and it is difficult to answer our second question and say which among them is the best abstraction. Nevertheless, if the algorithm is carried out for longer interval then it may be possible to distinguish the best abstraction.

Our third question was about the randomization in action selection and from the results, we found that the change in the degree of randomization of action selection will change the learning behavior. Hence the performance of learning will increase if action selection and other learning parameters are defined in a clever way.

To summarize, we meet our objective of improving the connection quality of terminals and the results answer our questions we had prior to the experiment.

Chapter 6

Discussion

In Chapter 4 we explained a learning algorithm for configuring the parameters of base stations. In this chapter, we will discuss what are the issues with our learning method and how one could deal with these issues in the future. We will also discuss other possible extensions and variations that can be done.

6.1 Factors Affecting Learning

The algorithm we used for the thesis work is a basic one and it focused on adjusting TXP settings according to the changing environment. The results in the previous chapter showed that the learning algorithm is effective and outperforms the base-line model. Some of the important factors that affect the learning are as follows.

Method of abstraction The design space of abstractions is large and consists of multiple orthogonal choices such as partitioning CQI vector and threshold values in abstraction of the partitions of the CQI vector according to the values in the vector components.

Score Computation The next important component in our learning algorithm is the scoring function. We defined weights for each CQI class in order to compute the score of states. Terminals in lower classes are more important in improving network behavior, thus we defined the weight vector accordingly. Other weight vectors could be suggested by network operators, for example if the connection quality could be quantified in financial terms.

Randomization and Learning Rate The randomization in choosing actions plays another important role in learning. More randomization

means lower rewards in the beginning which may cost more to the system at the initial stages, but in the long run, the algorithm increases rewards. On the other hand, less randomization could perform well in the beginning but with slower learning. Therefore, finding a balance between the randomization in choosing actions and the cost consideration is an important issue. In addition, tuning the learning parameters is equally important. Learning parameters can be adjusted according to what is the expected timescale of changes in the network environment or usage patterns. For example, a low learning rate would adapt to changes that take place over months, and a high learning rate would correspond to changes over days or weeks.

6.2 Possible Variations and Prospects for Future Work

There is room for improvement within our learning algorithm and many possible variations can be integrated as well. Below we discuss some other possible approaches and extensions that can be useful in the learning process.

Joint/Cooperative Apart from a limited form of independent learning, we also tried to perform joint learning by considering the control state of neighboring cells. We include the TXP values of neighboring cells in the state space and the actions are taken based on the joint reward. However, the method is not scalable and results in very large state spaces as the number of parameters increased when the parameters of other base stations are also considered. Moreover, the scores we obtained from joint learning were similar to the one from independent learning. However, one can still integrate the parameters of few neighboring cells to define the state space in a clever way so that there is a balance between state space size and learning performance. There are some scenarios where such joint learning approach can be useful. For example, if a cell in then network is dead (switched off), its neighboring cells can adjust their parameters to reconnect terminals previously connected to the now inactive base station. Furthermore, the problem of interference among the cells in the network can be better solved when the neighboring cells are taken into account when defining the state space.

Partial Observability The algorithm considered in this thesis work views the network management problem as a fully observable problem. However, in the mobile network, part of the current state of the system (location

and movement of mobile terminals) is not directly observable. Standard RL methods assume full observability. So our method essentially views a partially observable problem as fully observable. The implications of this for the learning approach need to be evaluated thoroughly. Consider a base station changing its antenna settings to achieve better QoS. The learned policy predicts a certain improvement, but the reality may differ from this, and if QoS actually got worse, the next action could be to restore the antenna settings, leading back to the starting state. With standard fully-observable RL methods this leads to oscillation between two control states, because nothing is remembered from the previous state of the system. The underlying problem is that the RL method views the system as fully observable with partial observability inaccurately (incorrectly) represented as nondeterminism/stochasticity of the control actions. In the (inaccurate) fully observable representation it makes sense to repeatedly try and fail with the control action that seemingly would (at least sometimes) lead to the better antenna settings, whereas with the (more accurate) partially observable model it would be recognized that the state of the system differs from the prediction, and that the system state is not precisely captured by the observations alone. Extensions of RL methods to partial observability have been proposed before. Jaakkola et al. [18] handled learning with partial observability by evaluating POMDP policy candidates with a Monte-Carlo simulation. Crites and Barto [13] apply RL to elevator control, a problem that has features not addressed by basic RL methods, including partial observability and multiple agents.

Increased complexity brought in by partial observability decreases the scalability of the learning methods, which is why finding a balance between accuracy of the models and the computational scalability becomes an important issue and this can be one of the extensions of the algorithm in the future.

Possible extension with current learning approach Apart from re-designing the algorithm, one could also do a simple extension in the current approach. In this work, the learning algorithm only takes the TXP settings for defining the state and actions, but one could also integrate other network parameters such as antenna tilt to the definition of the state space and actions. Integrating more parameters will better distinguish the state of the network but on the other hand, will increase the size of state space and the Q-value table. As a result, learning process slows down. Similarly, one could integrate RLF (Radio Link Failures) values in the algorithm with the objective of decreasing the number of lost connections.

Chapter 7

Conclusions

Mobile data traffic is expected to grow more than 500-fold between 2010 and 2020 [30]. Due to this, the upcoming 5G networks will have high demands in terms of speed and connection quality. In addition, there will be a lot more cells in 5G due to adoption of small cells technology. As a result, manual configuration of the network becomes infeasible and there will be a need for an automated approach so that the network can adapt itself to the highly changing environment. We presented a multi-agent reinforcement learning method that automates a part of network management in an efficient way and mitigates the complexity of managing the network in the future. Reinforcement learning is an approach to adaptive control in which learning and control processes are interleaved.

We defined the learning algorithm based on the Q-learning framework. The proposed algorithm views the network as a fully observable system with stochastic transitions from one state to another. The algorithm is composed of components such as state abstraction, exploration policy, and scoring function. The network parameters and the status of the terminals indicate the state of each cell in the network. However, considering all such information will result in very large state spaces. Therefore, we abstracted the state space to make it manageable. We used CQI vectors, which summarize terminals' connection quality, for defining the state space, and our abstraction method involves partitioning the CQI vector and mapping each partition into a value. Next, we defined the exploration function so that the algorithm randomizes more in the beginning of learning.

In reinforcement learning, the agent learns from the feedback of the environment. We considered three different ways of computing scoring functions which represent the feedback. The scores are based on the unabstracted and abstracted CQI vector. In addition, we defined a weight vector for both the CQI vector and its abstracted form.

We performed experiments with different methods of score computation by varying the abstracted values, components, and exploration strategies. All our experiments are conducted in a simulated environment. The results show that the learning is effective and performs better than the base-line model. The scores are increasing as well as there is an improvement in the connection quality of the terminals.

In this work, we used the learning algorithm to adjust the TXP values of the base stations. Nevertheless, one can also adjust other network parameters such as antenna tilt and integrate other key performance indicators such as RLF using the same approach. Apart from this, there is room for extension in many possible ways in the future. Learning with partial observability, cooperative learning that considers the neighboring base stations are some of the possible approaches that can be implemented in the near future and we hope the integration of these extensions in our current approach help to solve the network management problem even more effectively.

Bibliography

[1] Andre, D., and Russell, S. J. State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI* (2002), pp. 119–125.

[2] Atkeson, C. G., and Santamaria, J. C. A comparison of direct and model-based reinforcement learning. In *In International Conference on Robotics and Automation* (1997), IEEE Press, pp. 3557–3564.

[3] Baker, M., Sesia, S., and Toufik, I. *LTE-The UMTS Long Term Evolution From Theory to Practice*. Chichester: John Wiley & Sons, 2011.

[4] Barto, A. G., Bradtke, S. J., and Singh, S. P. Learning to act using real-time dynamic programming. *Artificial Intelligence 72*, 1 (1995), 81–138.

[5] Bellman, R. E. *Adaptive control processes: a guided tour*. Princeton University Press, 2015.

[6] Bennis, M., and Niyato, D. A Q-learning based approach to interference avoidance in self-organized femtocell networks. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE* (2010), pp. 706–710.

[7] Bennis, M., Perlaza, S. M., Blasco, P., Han, Z., and Poor, H. V. Self-organization in small cell networks: A reinforcement learning approach. *Wireless Communications, IEEE Transactions on 12*, 7 (2013), 3202–3212.

[8] Bernardo, F., Agustí, R., Pérez-Romero, J., and Sallent, O. Distributed spectrum management based on reinforcement learning. In *Cognitive Radio Oriented Wireless Networks and Communications, 2009. CROWNCOM'09.* (2009), pp. 1–6.

[9] Brafman, R. I., and Tennenholtz, M. R-max - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research 3*, Oct (2002), 213–231.

[10] Chernogorov, F., Repo, I., Räisänen, V., Nihtila, T., and Kurjenniemi, J. Cognitive self-healing system for future mobile networks. In *Wireless Communications and Mobile Computing Conference (IWCMC)* (2015), IEEE, pp. 628–633.

[11] Claus, C., and Boutilier, C. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI* (1998), pp. 746–752.

[12] Cox, C. *An introduction to LTE: LTE, LTE-advanced, SAE and 4G mobile communications*. John Wiley & Sons, 2012.

[13] Crites, R., and Barto, A. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8* (1996), pp. 1017–1023.

[14] Diuk, C., Li, L., and Leffler, B. R. The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (2009), ACM, pp. 249–256.

[15] Galindo-Serrano, A., Giupponi, L., and Auer, G. Distributed femto-to-macro interference management in multiuser ofdma networks. In *Proc. of IEEE 73rd Vehicular Technology Conference (VTC2011-Spring), Workshop on Broadband Femtocell Technologies, Budapest, Hungary* (2011), pp. 15–18.

[16] Hämäläinen, S., Sanneck, H., and Sartori, C. *LTE self-organising networks (SON): network management automation for operational efficiency*. John Wiley & Sons, 2012.

[17] Hätönen, K. *Data mining for telecommunications network log analysis*. Department of Computer Science, series of publications A, report A-2009-1. University of Helsinki, 2009.

[18] Jaakkola, T., Singh, S. P., and Jordan, M. I. Reinforcement learning algorithm for partially observable Markov decision problems. *Proceedings of the Advances in Neural Information Processing Systems* (1995), 345–352.

[19] Jaksch, T., Ortner, R., and Auer, P. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research 11*, Apr (2010), 1563–1600.

[20] Kaelbling, L. P. *Learning in embedded systems*. The MIT Press, 1993.

[21] Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4* (1996), 237–285.

[22] Kapetanakis, S., and Kudenko, D. Reinforcement learning of coordination in cooperative multi-agent systems. *AAAI/IAAI 2002* (2002), 326–331.

[23] Katehakis, M. N., and Veinott Jr, A. F. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research 12*, 2 (1987), 262–268.

[24] Kemptner, T., and Tsvetkov, B. T. LTE SON-Function Coordination Concept. *Network 101* (2013).

[25] Kumpulainen, P. Anomaly detection for communication network monitoring applications. *Tampere University of Technology. Publication; 1192* (2014).

[26] Lauer, M., and Riedmiller, M. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the Seventeenth International Conference on Machine Learning* (2000), pp. 535–542.

[27] Lee, M., Marconett, D., Ye, X., and Yoo, S. B. Cognitive network management with reinforcement learning for wireless mesh networks. In *International Workshop on IP Operations and Management* (2007), Springer, pp. 168–179.

[28] Li, H. Multi-agent Q-learning of channel selection in multi-user cognitive radio systems: a two by two case. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on* (2009), IEEE, pp. 1893–1898.

[29] Littman, M. L. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning* (1994), vol. 157, pp. 157–163.

[30] Nakamura, T., Nagata, S., Benjebbour, A., Kishiyama, Y., Hai, T., Xiaodong, S., Ning, Y., and Nan, L. Trends in small cell enhancements in LTE advanced. *IEEE Communications Magazine 51*, 2 (2013), 98–105.

[31] Neto, G. From single-agent to multi-agent reinforcement learning: Foundational concepts and methods. Technical report, Learning Theory Course, Instituto de Sistemas e Robótica, Instituto Superior Técnico, Lisbon, 2005.

[32] Niyato, D., and Hossain, E. Dynamics of network selection in heterogeneous wireless networks: an evolutionary game approach. *IEEE transactions on vehicular technology 58*, 4 (2009), 2008–2017.

[33] Østerbø, O., and Grøndalen, O. Benefits of self-organizing networks (SON) for mobile operators. *Journal of Computer Networks and Communications 2012* (2012).

[34] Räisänen, V., and Tang, H. Knowledge modeling for conflict detection in self-organized networks. In *Mobile Networks and Management*. Springer, 2011, pp. 107–119.

[35] Razavi, R., Klein, S., and Claussen, H. Self-optimization of capacity and coverage in LTE networks using a fuzzy reinforcement learning approach. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2010 IEEE 21st International Symposium on* (2010), IEEE, pp. 1865–1870.

[36] Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 2009.

[37] Seijen, H., Whiteson, S., and Kester, L. Efficient abstraction selection in reinforcement learning. *Computational Intelligence 30*, 4 (2014), 657–699.

[38] Sutton, R. S., and Barto, A. G. *Reinforcement learning: An introduction*. MIT Press, 1998.

[39] Suutarinen, J. Performance measurements of GSM base station system. *Licentiate's thesis, Tampere University of Technology, Finland* (1994).

[40] Szilágyi, P., and Nováczki, S. An automatic detection and diagnosis framework for mobile communication systems. *Network and Service Management, IEEE Transactions on 9*, 2 (2012), 184–197.

[41] Tan, M. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning* (1993), pp. 330–337.

[42] Thrun, S. The role of exploration in learning control. In *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, D. White and D. Sofge, Eds. Van Nostrand Reinhold, 1992.

[43] Thrun, S. B. Efficient exploration in reinforcement learning. Technical report CMU-CS-9 2-102, Carnegie Mellon University, 1992.

[44] Tokic, M. Adaptive ε-greedy exploration in reinforcement learning based on value differences. In *Annual Conference on Artificial Intelligence* (2010), Springer, pp. 203–210.

[45] Tokic, M., and Palm, G. Value-difference based exploration: Adaptive control between epsilon-greedy and softmax. In *KI* (2011), Springer, pp. 335–346.

[46] Watkins, C. J., and Dayan, P. Q-learning. *Machine Learning 8*, 3-4 (1992), 279–292.

[47] Watkins, C. J. C. H. *Learning from delayed rewards*. PhD thesis, University of Cambridge, England, 1989.

[48] Weiss, G. Distributed reinforcement learning. *Robotics and Autonomous Systems 15* (1995), 135–142.

[49] Wiering, M. Multi-agent reinforcement learning for traffic light control. In *ICML* (2000), pp. 1151–1158.